Design and Analysis of Algorithms
Dynamic Programming

Si Wu

School of CSE, SCUT

cswusi@scut.edu.cn

TA: Wenhao Wu (1565865638@qq.com)

Yi Liu (1337545838@qq.com)

Topics

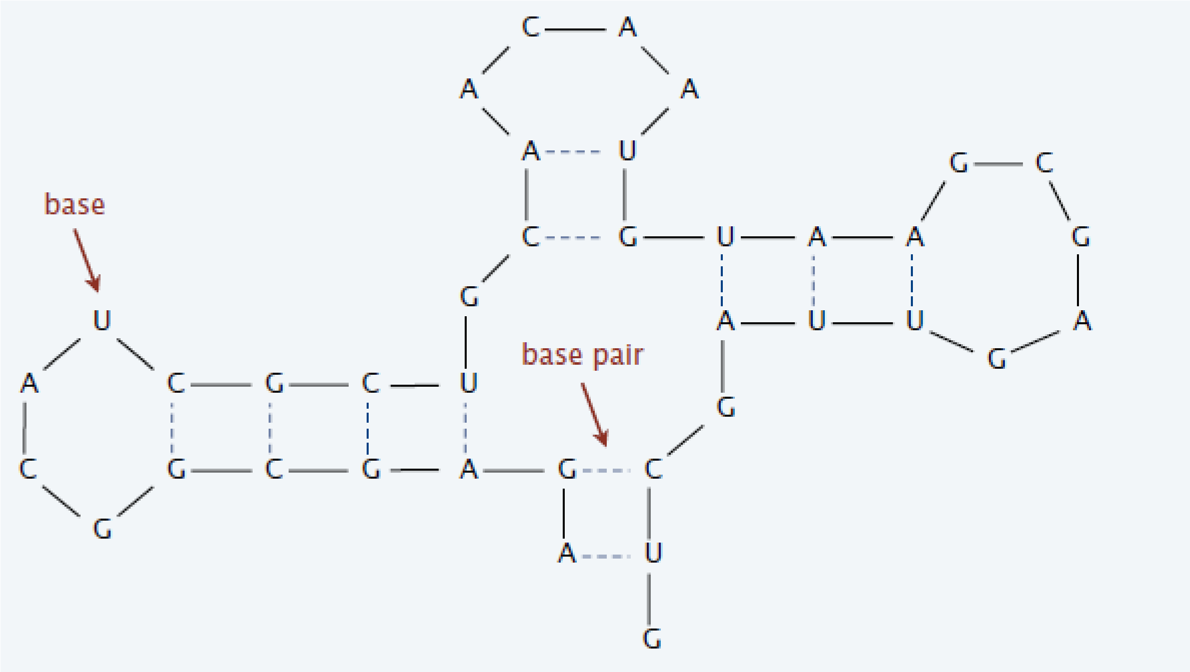
- **RNA Secondary Structure**
- **Sequence Alignment**
- **Hirschberg's Algorithm**
- **Bellman-Ford Algorithm**

RNA Secondary Structure

RNA. String $B = b_1 b_2 \ldots b_n$ over alphabet {A, C, G, U}.

Secondary structure. RNA is single-stranded so it tends to loop back and form base pairs with itself. This structure is essential for understanding behavior of molecule.

https://rna.urmc.rochester.edu/RNAstructureWeb/Servers/Predict1/Example.php



RNA secondary structure for GUCGAUUGAGCGAAUGUAACAACGUGGCUACGGCGAGA

RNA Secondary Structure

Secondary structure. A set of pairs $S = \{(b_i, b_j)\}$ that satisfy:

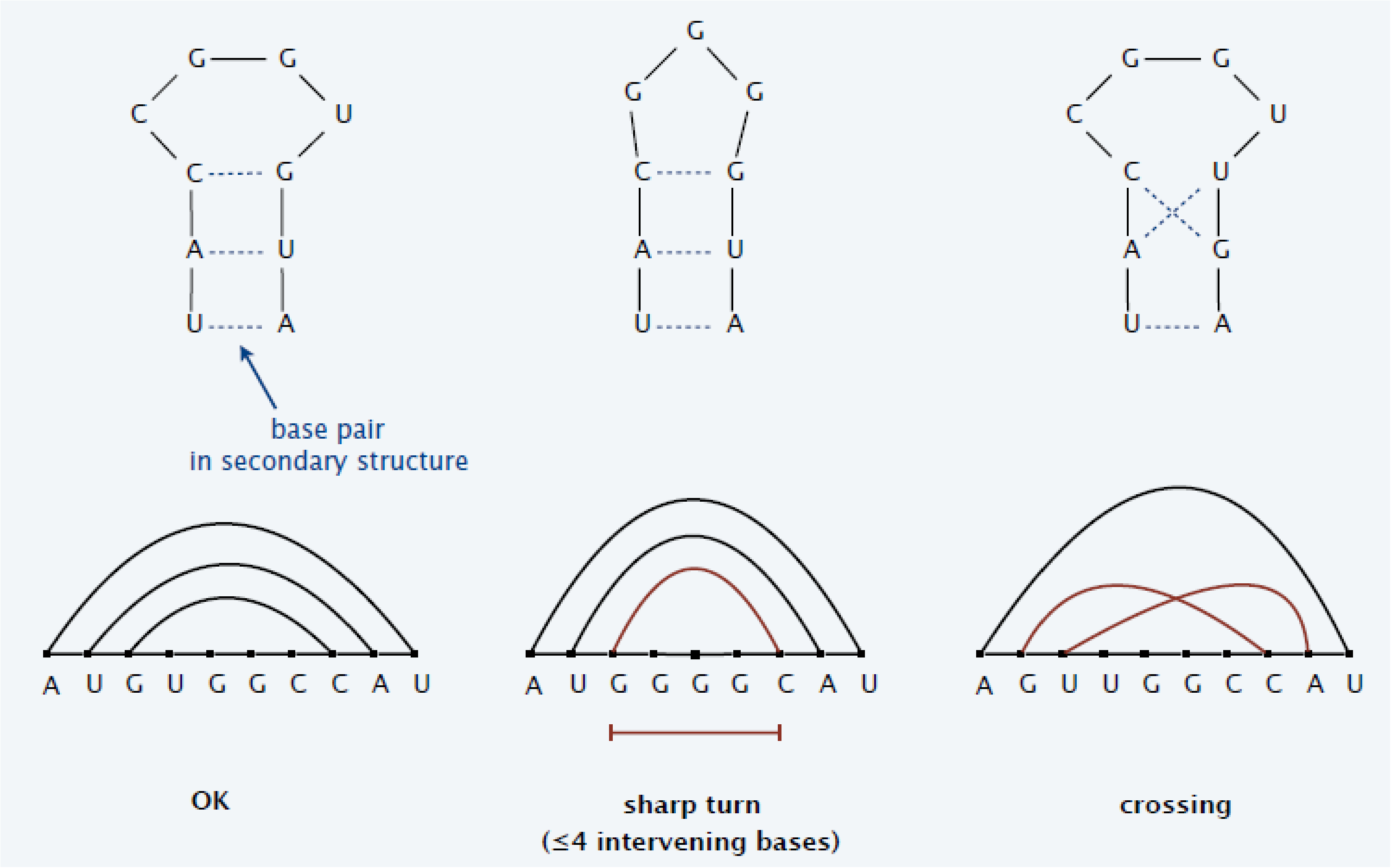
- Each pair in S is a Watson-Crick complement: A-U, U-A, C-G, or G-C.
- The ends of each pair are separated by at least 4 intervening bases. If $(b_i, b_j) \in S$, then $i < j - 4$.
- If (b_i, b_j) and (b_k, b_l) are two pairs in S, then we cannot have $i < k < j < l$.

Free energy. Usual hypothesis is that an RNA molecule will form the secondary structure with the minimum total free energy. (approximate by the number of base pairs)

Goal. Given an RNA molecule $B = b_1 b_2 \ldots b_n$, find a secondary structure S that maximizes the number of base pairs.

RNA Secondary Structure

Examples.

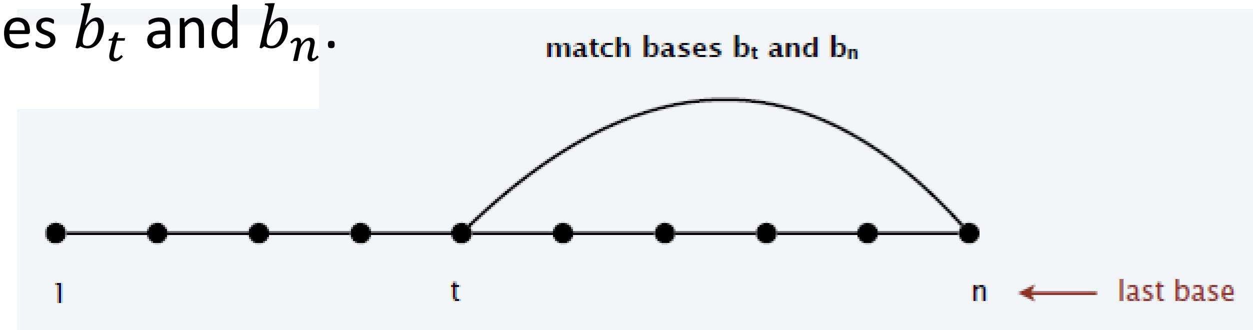


RNA Secondary Structure: Sub-problems

First attempt. $OPT(j)$ = maximum number of base pairs in a secondary of the substring $b_1 b_2 \ldots b_j$.

Goal. $OPT(n)$

Choice. Match bases b_t and b_n.



Difficulty. Results in two sub-problems (but one of wrong form).

- Find secondary structure in $b_1 b_2 \ldots b_{t-1}$. $(OPT(t-1))$
- Find secondary structure in $b_{t+1} b_2 \ldots b_{n-1}$. (need more sub-problems)

Dynamic Programming Over Intervals

Notation. $OPT(i, j)$ = maximum number of base pairs in a secondary of the substring $b_i b_{i+1} \dots b_j$.

Case 1. If $i \geq j - 4$.

- $OPT(i, j) = 0$ by no-sharp turns condition.

Case 2. Bases b_j is not involved in a pair.

- $OPT(i, j) = OPT(i, j - 1)$.

Case 3. Bases b_j pairs with b_t for some $i \leq t < j - 4$.

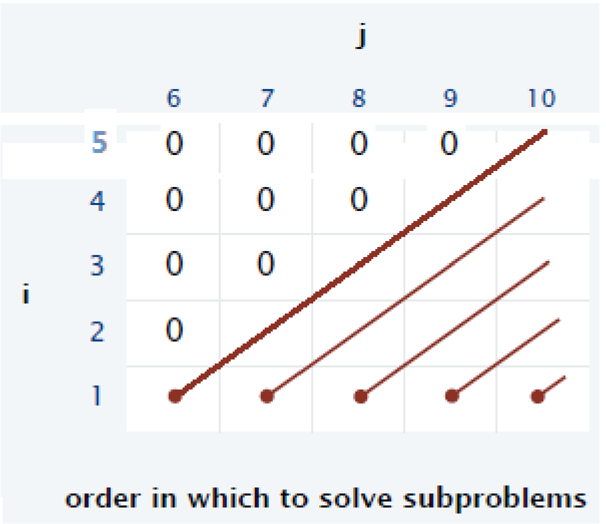
- Non-crossing constraint decouples resulting sub-problems.
- $OPT(i, j) = 1 + \max_t \{OPT(i, t - 1) + OPT(t + 1, j - 1)\}$.

(take max over t such that $i \leq t < j - 4$, b_t and b_j are Watson-Crick complements)

Bottom-Up Dynamic Programming Over Intervals

Q. In which order to solve the sub-problems?

A. Do shortest intervals first.



RNA-Secondary-Structure $(n, b_1, b_2, \dots, b_n)$

For $k = 5$ To $n - 1$

  For $i = 1$ To $n - k$

    $j \leftarrow i + k$.

    For each b_t $(i \le t < j - 4)$ paired with b_j

      $T = 1 + M[i, t - 1] + M[t + 1, j - 1]$.

      $M[i, j] \leftarrow \max\{M[i, j - 1], T\}$.

Return $M[1, n]$.

RNA Secondary Structure: An Example

RNA sequence. A C C G G U A G U

A	C	C	G	G	U	A	G	U
1	2	3	4	5	6	7	8	9

4	0	0	0	
3	0	0		
2	0			
$i = 1$				
	$j = 6$	7	8	9

Initial values

RNA-Secondary-Structure $(n, b_1, b_2, \ldots, b_n)$

For $k = 5$ To $n - 1$

  For $i = 1$ To $n - k$

    $j \leftarrow i + k.$

    For each b_t $(i \le t < j - 4)$ paired with b_j

      $T = 1 + M[i, t - 1] + M[t + 1, j - 1].$

      $M[i, j] \leftarrow \max\{M[i, j - 1], T\}.$

Return $M[1, n]$.

RNA Secondary Structure: An Example

RNA sequence. A C C G G U A G U

A	C	C	G	G	U	A	G	U
1	2	3	4	5	6	7	8	9

$$\boxed{i \le t < j - 4}$$

	$j = 6$	7	8	9
4	0	0	0	
3	0	0		
2	0			
$i = 1$				

Initial values

	$j = 6$	7	8	9
4	0	0	0	0
3	0	0	1	
2	0	0		
$i = 1$	1			

Filling in the values for $k = 5$

	$j = 6$	7	8	9
4	0	0	0	0
3	0	0	1	1
2	0	0	1	
$i = 1$	1	1		

Filling in the values for $k = 6$

	$j = 6$	7	8	9
4	0	0	0	0
3	0	0	1	1
2	0	0	1	1
$i = 1$	1	1	1	

Filling in the values for $k = 7$

	$j = 6$	7	8	9
4	0	0	0	0
3	0	0	1	1
2	0	0	1	1
$i = 1$	1	1	1	2

Filling in the values for $k = 8$

String Similarity

Q. How similar are two strings?

Ex. ocurrance & occurrence.

o	c	u	r	r	a	n	c	e	–
o	c	c	u	r	r	e	n	c	e

6 mismatches, 1 gap

o	c	–	u	r	r	a	n	c	e
o	c	c	u	r	r	e	n	c	e

1 mismatch, 1 gap

o	c	–	u	r	r	–	a	n	c	e
o	c	c	u	r	r	e	–	n	c	e

0 mismatches, 3 gaps

Edit Distance

Edit distance.

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

C	T	–	G	A	C	C	T	A	C	G
C	T	G	G	A	C	G	A	A	C	G

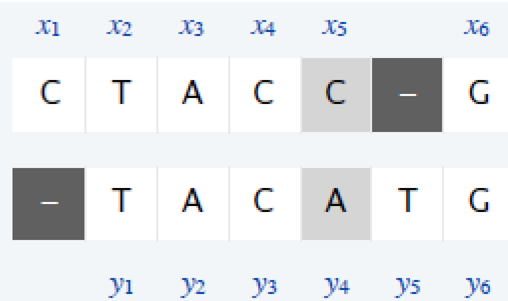
$$\text{cost} = \delta + \alpha_{CG} + \alpha_{TA}$$

Applications. Speech recognition, computational biology,…

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 item occurs in at most one pair and no crossings ($x_i - y_j$ and $x_h - y_k$ cross if $i < h$, but $j > k$).



an alignment of CTACCG and TACATG:

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

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 item occurs in at most one pair and no crossings ($x_i - y_j$ and $x_h - y_k$ cross if $i < h$, but $j > k$).

Def. The cost of an alignment M is:

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

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. $OPT(i, j)$ includes $x_i - y_j$.
Pay mismatch for $x_i - y_j$ + min cost of aligning $x_1 x_2 \ldots x_{i-1}$ and $y_1 y_2 \ldots y_{j-1}$.

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 $y_1 y_2 \ldots y_j$.

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 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 $y_1 y_2 \ldots y_{j-1}$.

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

Sequence Alignment: Bottom-Up 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
    $M[i, j] \leftarrow \min\{\alpha[x_i, y_j] + M[i-1, j-1],$
        $\delta + M[i-1, j], \delta + M[i, j-1]\}$.
Return $M[m, n]$.

Sequence Alignment: An Example

Ex. Align the words *mean* and *name*. Assume that $\delta = 2$; matching a vowel with a different vowel, or a consonant with a different consonant, costs 1; while matching a vowel, or a consonant with each other costs 3.

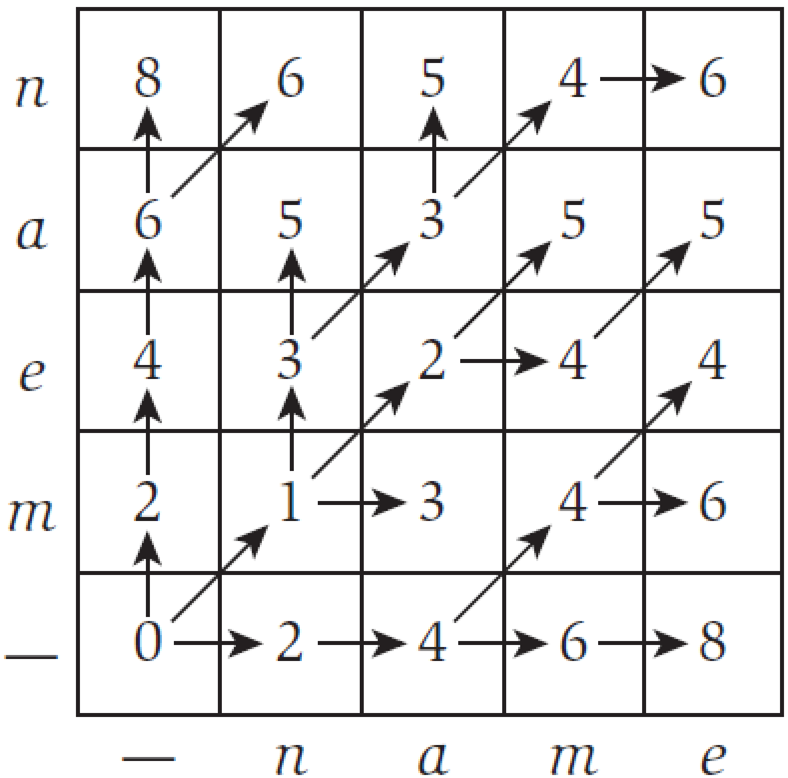
Sequence-Alignment $(m, n, x_1, \ldots, x_m, y_1, \ldots, y_n, \delta, \alpha)$

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
    $M[i, j] \leftarrow \min\{\alpha[x_i, y_j] + M[i-1, j-1],$
      $\delta + M[i-1, j], \delta + M[i, j-1]\}$.
Return $M[m, n]$.

n					
a					
e					
m					
-					
	-	**n**	**a**	**m**	**e**

Sequence Alignment: An Example

Ex. Align the words *mean* and *name*. Assume that $\delta = 2$; matching a vowel with a different vowel, or a consonant with a different consonant, costs 1; while matching a vowel, or a consonant with each other costs 3.

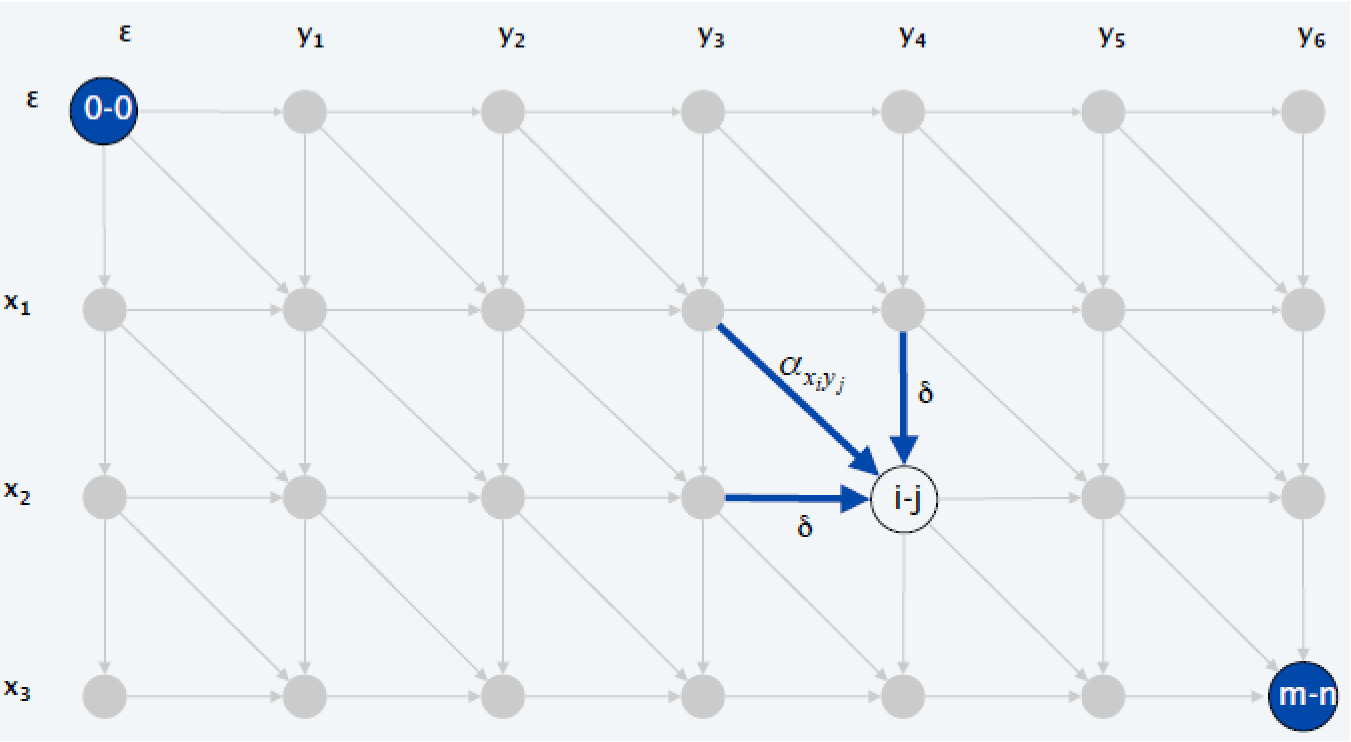


By following arrows backward from node (4,4), we can trace back to construct the alignment.

Hirschberg's Algorithm

Edit distance graph.

- Let $f(i, j)$ be shortest path from $(0,0)$ to (i, j).
- Lemma: $f(i, j) = OPT(i, j)$ for all i and j.



Hirschberg's Algorithm

Edit distance graph.

- Let $f(i,j)$ be shortest path from $(0,0)$ to (i,j).
- Lemma: $f(i,j) = OPT(i,j)$ for all i and j.

Pf of Lemma.

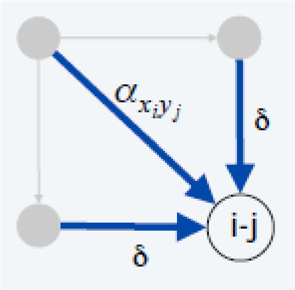
- Base case: $f(0,0) = OPT(0,0) = 0$.
- Inductive hypothesis: assume true for all (i',j') with $i' + j' < i + j$.
- Last edge on the shortest path to (i,j) is from $(i-1,j-1)$, $(i-1,j)$, or $(i,j-1)$.

Hirschberg's Algorithm

Pf of Lemma.

- Base case: $f(0,0) = OPT(0,0) = 0$.
- Inductive hypothesis: assume true for all (i', j') with $i' + j' < i + j$.
- Last edge on the shortest path to (i, j) is from $(i - 1, j - 1)$, $(i - 1, j)$, or $(i, j - 1)$.
- Thus,

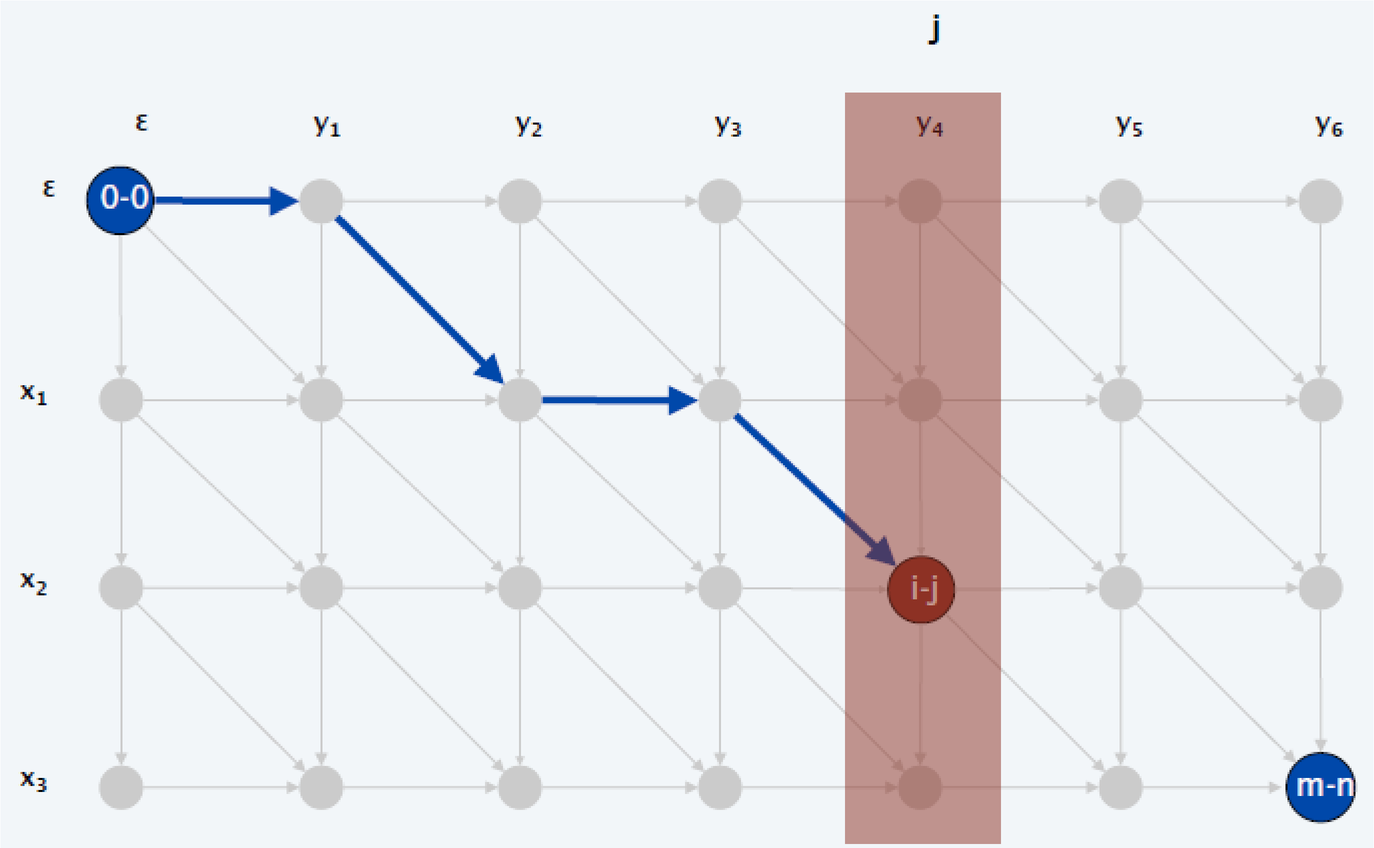
$$f(i,j) = \min\{\alpha_{x_i y_j} + f(i-1, j-1), \delta + f(i-1, j), \delta + f(i, j-1)\}$$
$$= \min\{\alpha_{x_i y_j} + OPT(i-1, j-1), \delta + OPT(i-1, j), \delta + OPT(i, j-1)\}$$
$$= OPT(i,j).$$



Hirschberg's Algorithm

Edit distance graph.

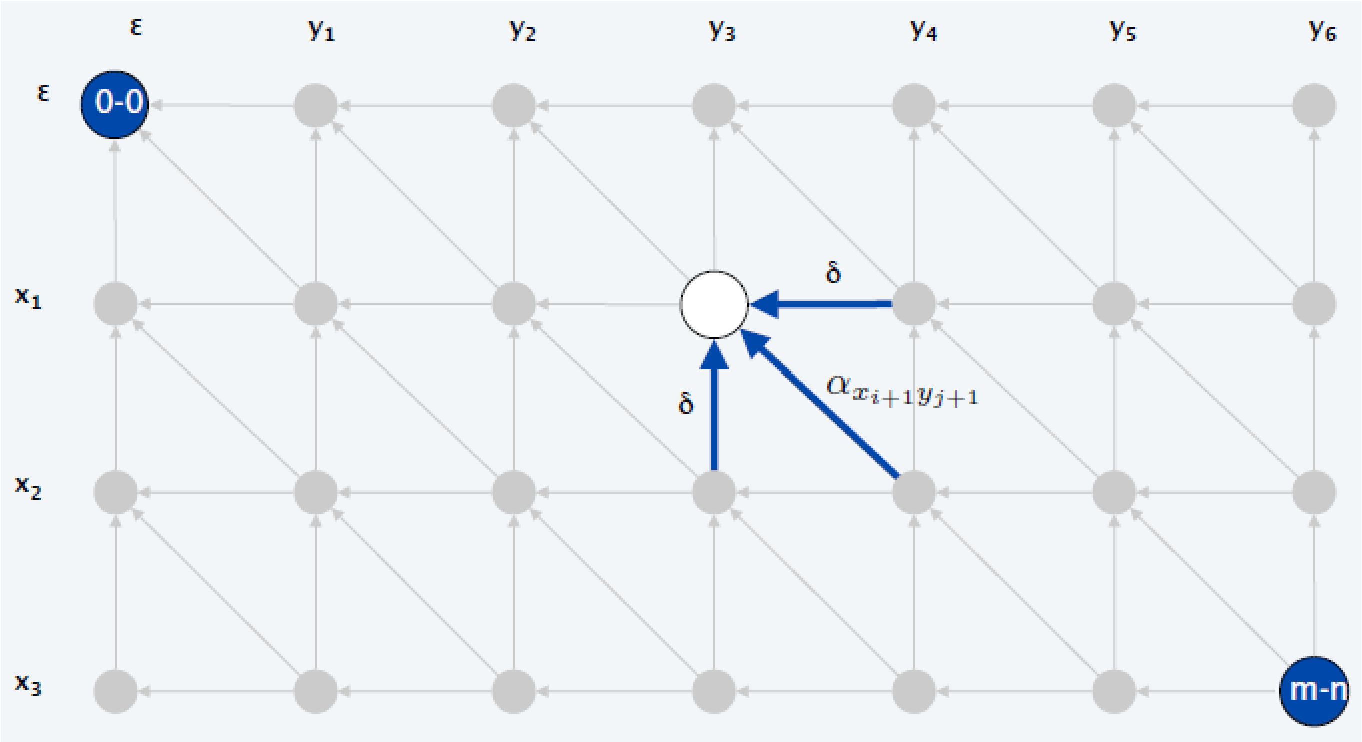
- Let $f(i, j)$ be shortest path from $(0,0)$ to (i, j).
- Lemma: $f(i, j) = OPT(i, j)$ for all i and j.
- Can compute $f(\cdot, j)$ for any j in $O(mn)$ time.



Hirschberg's Algorithm

Edit distance graph.

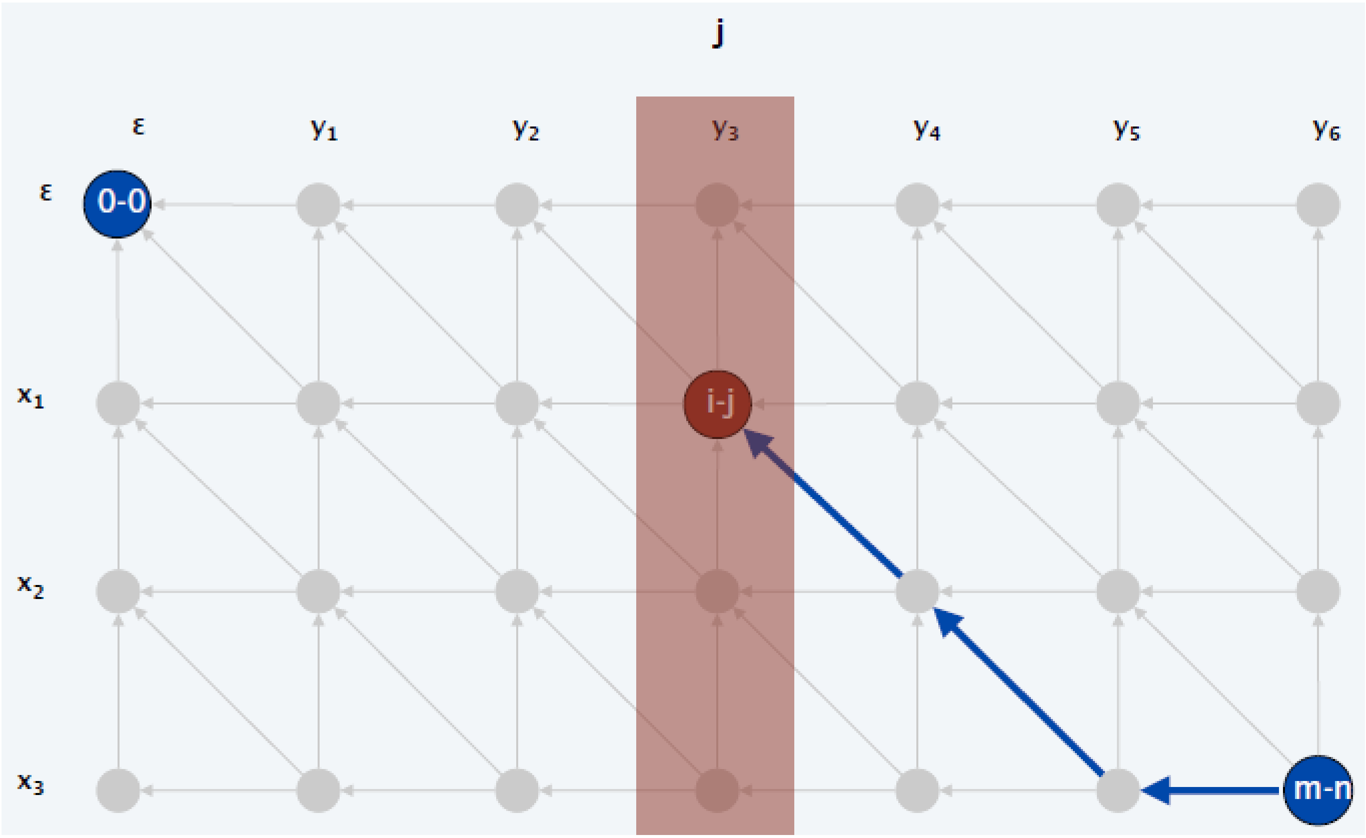
- Let $g(i,j)$ be shortest path from (i,j) to (m,n).
- Can compute by reversing the edge orientations and inverting the roles of $(0,0)$ to (m,n).



Hirschberg's Algorithm

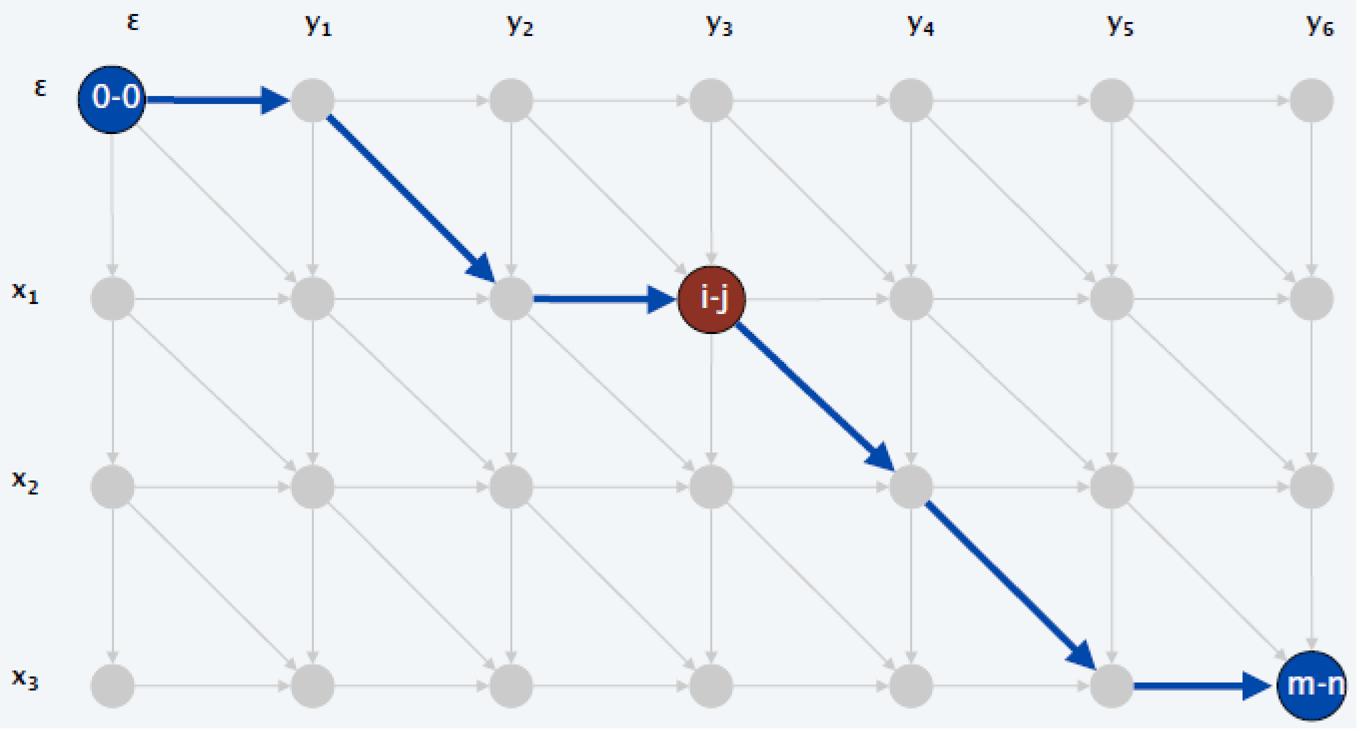
Edit distance graph.

- Let $g(i, j)$ be shortest path from (i, j) to (m, n).
- Can compute $g(\cdot, j)$ for any j in $O(mn)$ time.



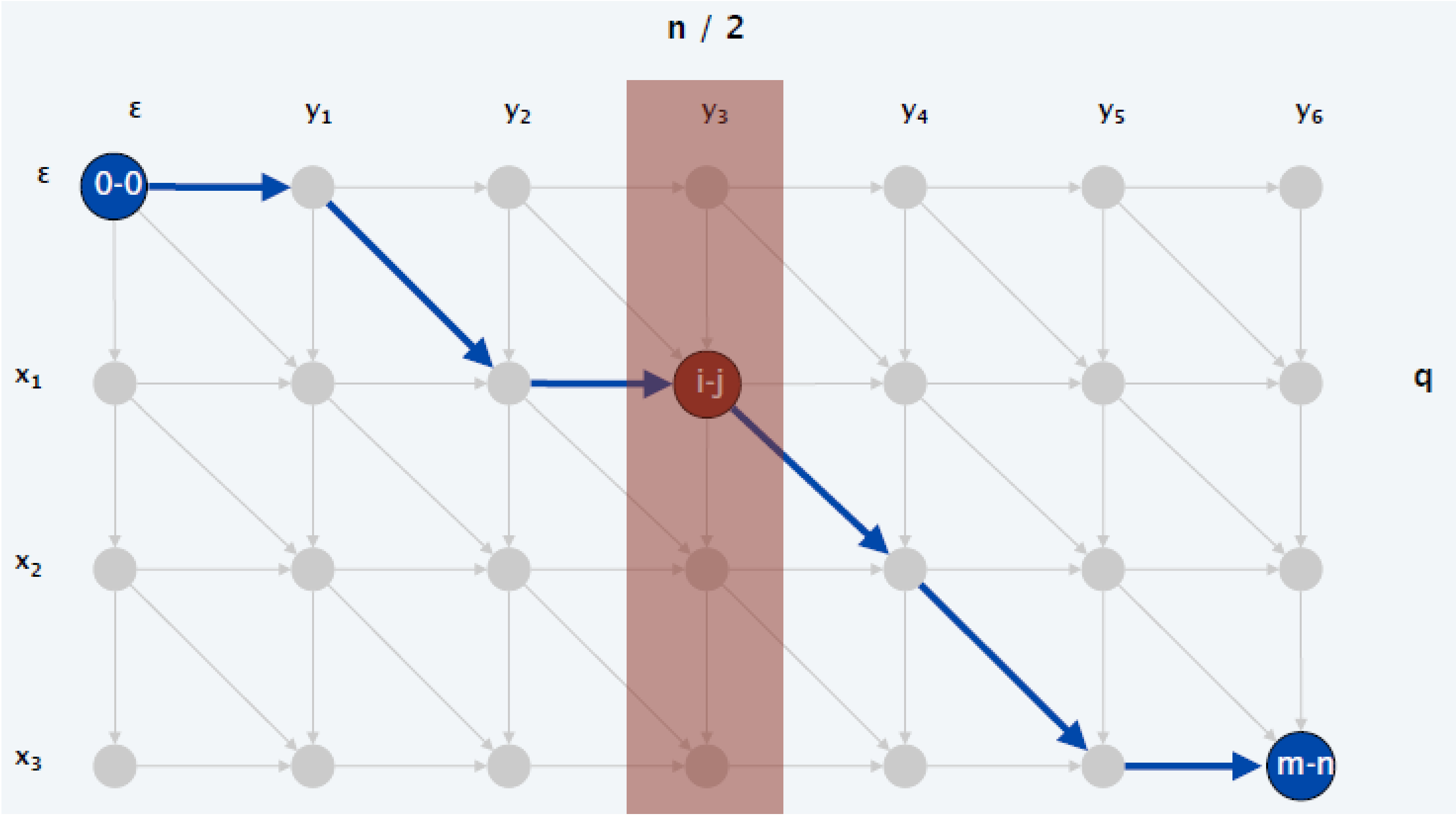
Hirschberg's Algorithm

Observation 1. The cost of a shortest path that uses (i, j) is $f(i, j) + g(i, j)$.



Hirschberg's Algorithm

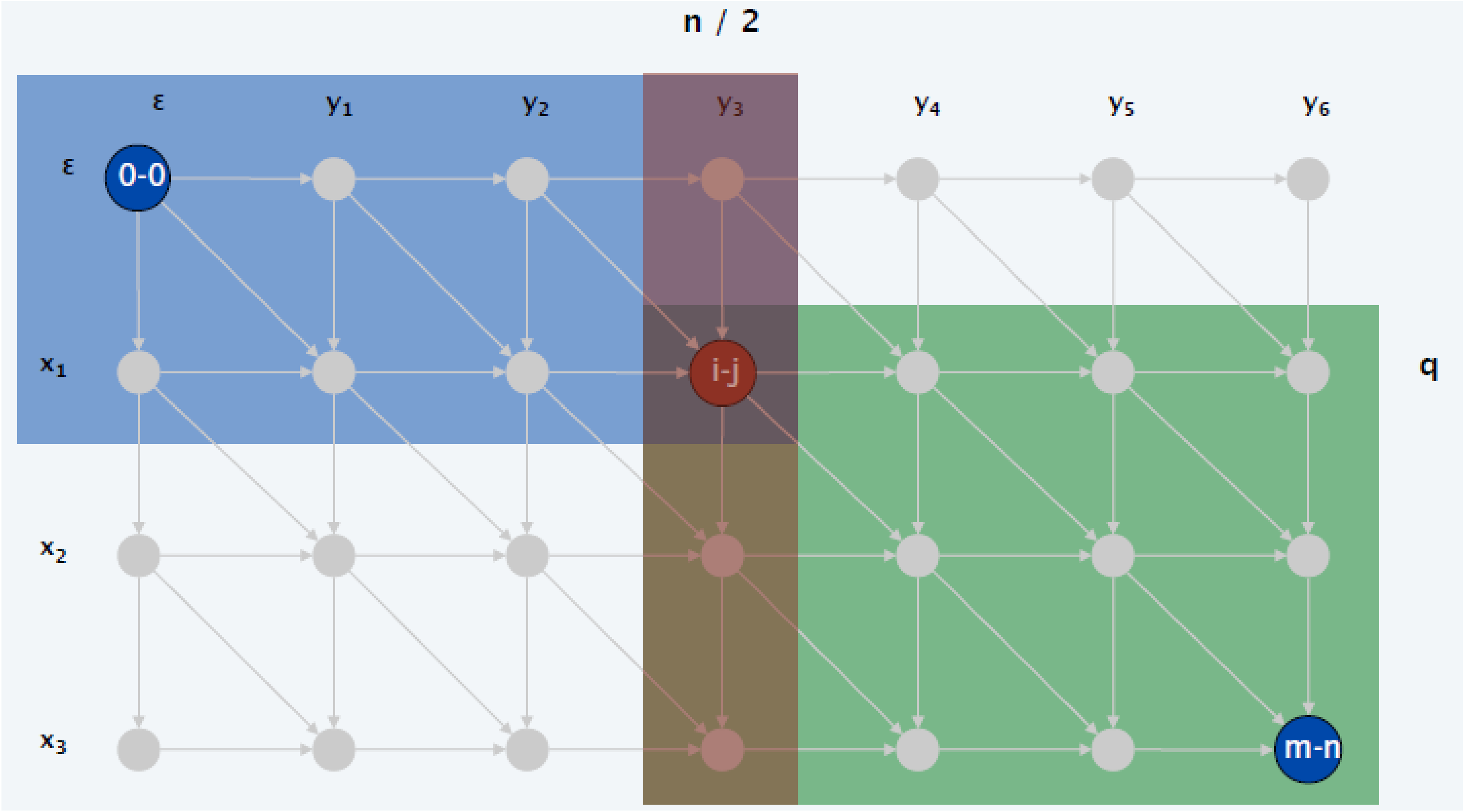
Observation 2. Let q be an index that minimizes $f(q, n/2) + g(q, n/2)$. Then, there exists a shortest path from $(0,0)$ to (m, n) that use $(q, n/2)$.



Hirschberg's Algorithm

Divide. Find index q that minimizes $f(q, n/2) + g(q, n/2)$; align x_q and $y_{n/2}$.

Conquer. Recursively compute optimal alignment in each piece.



Hirschberg's Algorithm: Running Time Analysis Warmup

Theorem. Let $T(m,n)$ = max running time of Hirschberg's algorithm on strings of lengths at most m and n. Then $T(m,n) = O(mnlogn)$.

Pf.

$$T(m,n) \le 2T\left(m, \frac{n}{2}\right) + O(mn)$$
$$\rightarrow T(m,n) = O(mnlogn)$$

Remark. Analysis is not tight because two sub-problems are of size $(q, \frac{n}{2})$ and $(m-q, \frac{n}{2})$. In next slide, we save $logn$ factor.

Hirschberg's Algorithm: Running Time Analysis

Theorem. Let $T(m,n) =$ max running time of Hirschberg's algorithm on strings of lengths at most m and n. Then $T(m,n) = O(mn)$.

Pf. [by induction on n]

- $O(mn)$ time to compute $f\left(\cdot,\frac{n}{2}\right)$ and $g\left(\cdot,\frac{n}{2}\right)$ and find index q.
- $T\left(q,\frac{n}{2}\right) + T\left(m-q,\frac{n}{2}\right)$ time for two recursive calls.
- Choose constant c so that: $T(m,2) \le cm, T(2,n) \le cn,$

$$T(m,n) \le cmn + T\left(q,\frac{n}{2}\right) + T\left(m-q,\frac{n}{2}\right)$$

- Claim. $T(m,n) \le 2cmn$.

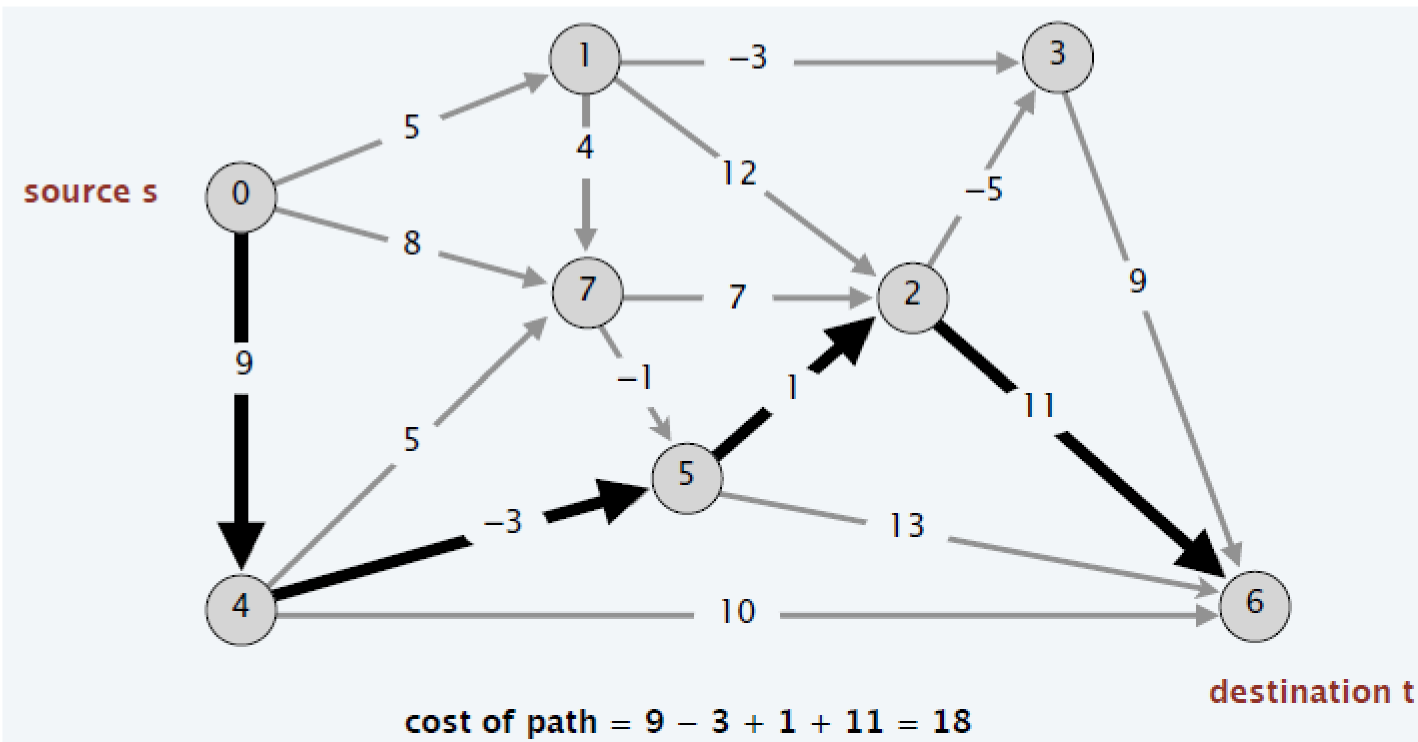
Hirschberg's Algorithm: Running Time Analysis

- Claim. $T(m,n) \leq 2cmn$.
- Base cases: $m = 2$ or $n = 2$.
- Inductive hypothesis: $T(m,n) \leq 2cmn$ for all (m', n') with $m' + n' < m + n$.

$$
\begin{aligned}
T(m,n) &\leq T\left(q, \frac{n}{2}\right) + T\left(m-q, \frac{n}{2}\right) + cmn \\
&\leq \frac{2cqn}{2} + \frac{2c(m-q)n}{2} + cmn \\
&= cqn + cmn - cqn + cmn \\
&= 2cmn.
\end{aligned}
$$

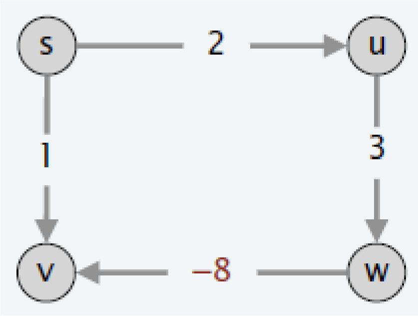
Shortest Paths

Shortest-path problem. Given a digraph $G = (V, E)$, with arbitrary edge weights or cost c_{vw}, find cheapest path from node s to node t.

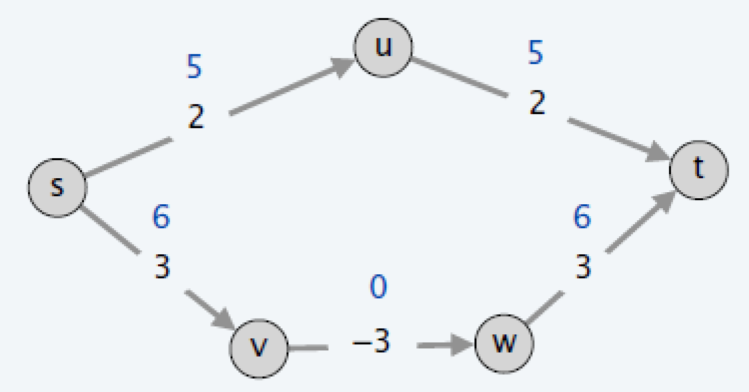


Shortest Paths: Failed Attempts

Dijkstra. May not produce shortest paths when edge weights are negatives.

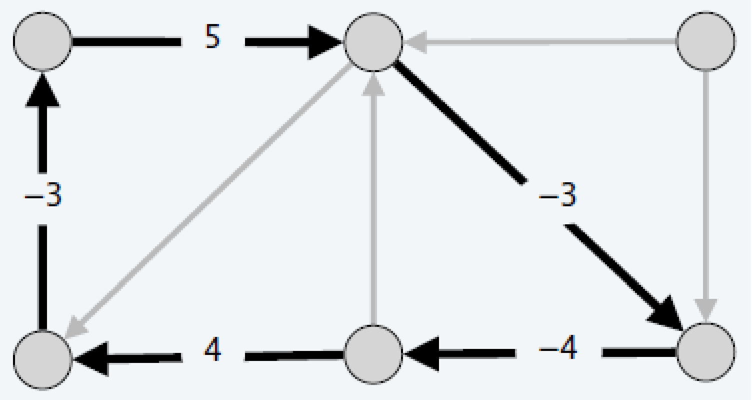


Reweighting. Adding a constant to every edge weight does not necessarily make Dijkstra’s algorithm produce shortest paths.



Negative Cycles

Def. A negative cycle is a directed cycle such that sum of its edge weight is negative.



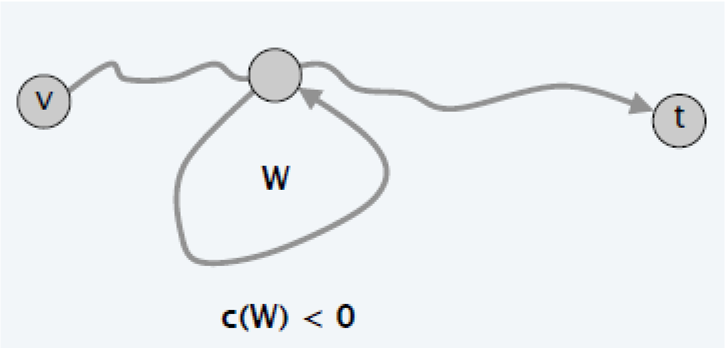
a negative cycle W : $c(W) = \sum_{e \in W} c_e < 0$

Shortest Paths and Negative Cycles

Lemma 1. If some path from v to t contains a negative cycle, then there does not exist a cheapest path from v to t.

Pf.

If there exists such a cycle W, then can build a $v \to t$ path of arbitrarily negative weight by detouring around cycle as many times as desired.

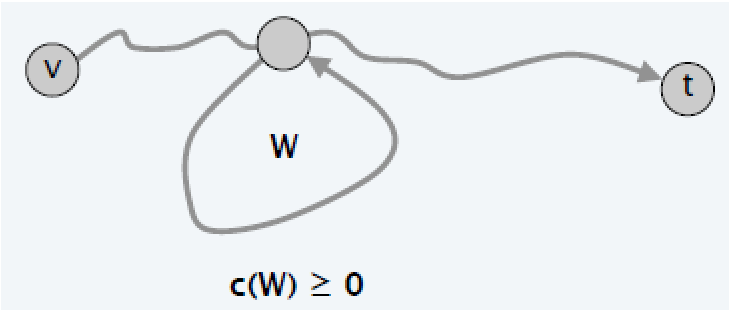


Shortest Paths and Negative Cycles

Lemma 2. If G has no negative cycles, then there exists a cheapest path from v to t that is simple (i.e. does not repeat nodes), and hence has at most $\leq n-1$ edges.

Pf.

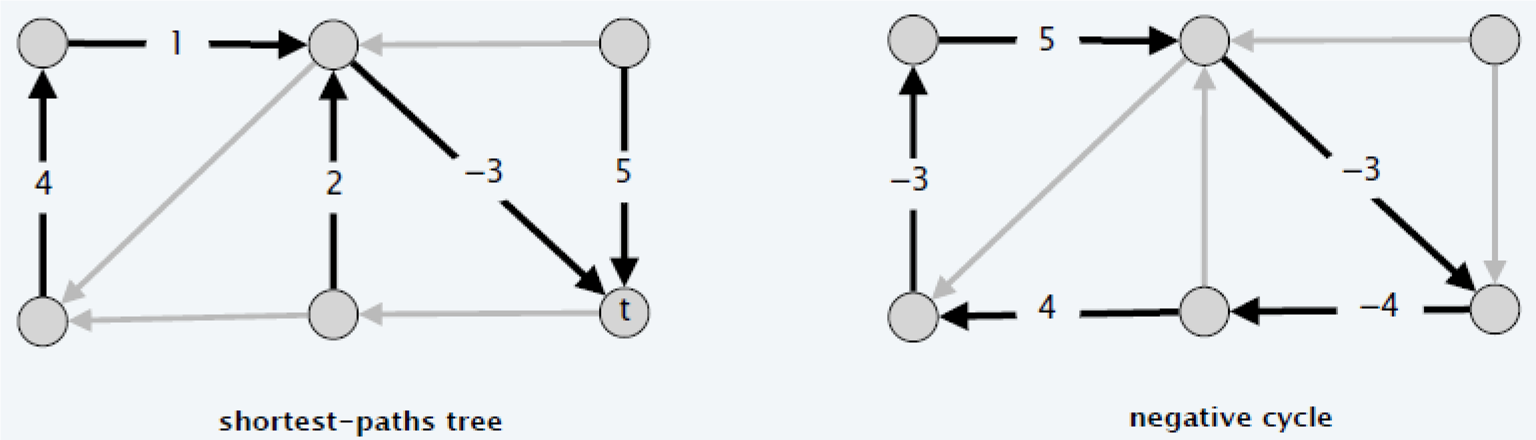
- Consider a cheapest $v \rightarrow t$ path P that uses the fewest edges.
- If P contains a cycle W, can remove portion of P corresponding to W without increasing the cost.



Shortest Paths and Negative-Cycles Problems

Single-destination shortest-paths problem. Given a digraph $\mathrm{G} = (V, E)$ with edge weights c_{vw}, and no negative cycles and a distinguished note t, find cheapest $v \rightarrow t$ path for each node v.

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



Shortest Paths: Dynamic Programming

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

- Case 1: Cheapest $v \to t$ path uses $\leq i - 1$ edges.
  - $OPT(i, v) = OPT(i - 1, v)$.
- Case 2: Cheapest $v \to t$ path uses exactly i edges.
  - If (v, w) is the first edge, then OPT uses (v, w), and then selects best $w \to t$ path using $\leq i - 1$ edges.

$$OPT(i, v) = \begin{cases} \infty & if\ i = 0 \\ \min\left\{OPT(i-1, v), \min\limits_{(v,w)\in E}\{OPT(i-1, w) + c_{vw}\}\right\} & otherwise \end{cases}$$

Observation. If no negative cycles, $OPT(n - 1, v)$ = cost of cheapest $v \to t$ path.

Shortest Paths: Implementation

Shortest-Paths (V, E, c, t)

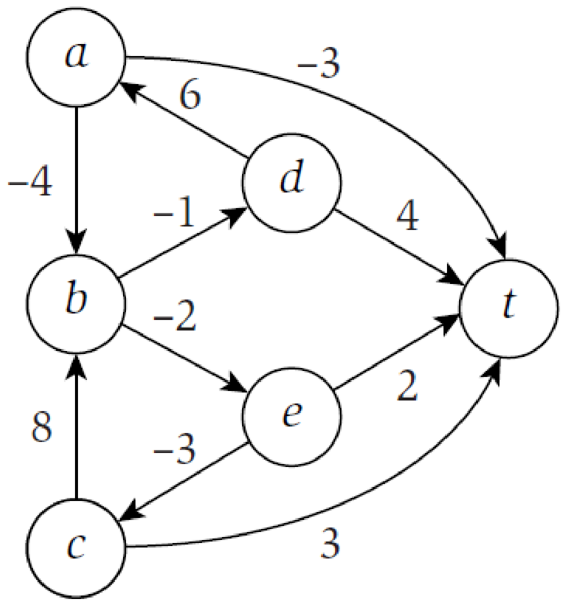
For each node $v \in V$
  $M[0, v] \leftarrow \infty.$

$M[0, t] \leftarrow 0.$

For $i = 0$ To $n - 1$
  For each node $v \in V$
    $M[i, v] \leftarrow M[i-1, v].$
    For each edge $(v, w) \in E$
      $M[i, v] \leftarrow \min\{M[i, v], M[i-1, w] + c_{vw}\}.$

Shortest Paths: An Example

Ex. Considering the following directed graph, find a shortest path from each node to t.



Shortest-Paths (V, E, c, t)

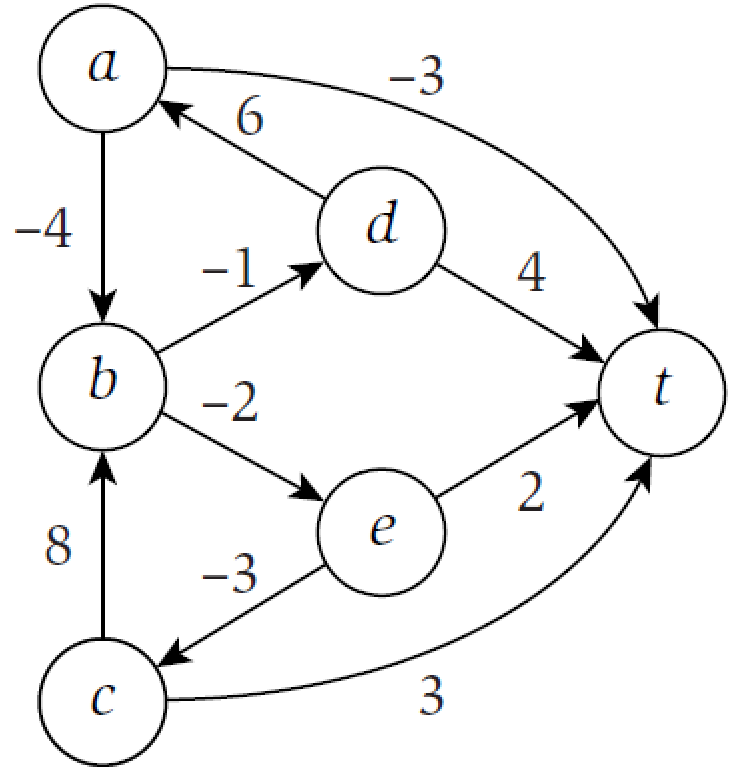
For each node $v \in V$
  $M[0, v] \leftarrow \infty.$
$M[0, t] \leftarrow 0.$
For $i = 0$ To $n - 1$
  For each node $v \in V$
    $M[i, v] \leftarrow M[i-1, v].$
    For each edge $(v, w) \in E$
      $M[i, v] \leftarrow \min\{M[i, v], M[i-1, w] + c_{vw}\}.$

	0	1	2	3	4	5
t						
a						
b						
c						
d						
e						

Shortest Paths: An Example

Ex. Considering the following directed graph, find a shortest path from each node to t.

	0	1	2	3	4	5
t	0	0	0	0	0	0
a	∞	−3	−3	−4	−6	−6
b	∞	∞	0	−2	−2	−2
c	∞	3	3	3	3	3
d	∞	4	3	3	2	0
e	∞	2	0	0	0	0



Each row corresponds to the shortest path from a node to t, as we allow the path to use an increasing number of edges

Shortest Paths: Implementation

Theorem 1. Given a digraph $G = (V, E)$ with no negative cycles, the dynamic programming algorithm computes the cost of a cheapest $v \to t$ path for each node v in $\Theta(mn)$ time.

Pf.

- Each iteration i takes $\Theta(m)$ time since we examine each edge once.

Finding the shortest paths.

- Approach 1: Maintain a $successor(i, v)$ that points to next node on cheapest $v \to t$ path using at most i edges.
- Approach 2: Compute optimal costs $M[i, v]$ and consider only edges with $M[i, v] = M[i-1, w] + c_{vw}$.

Shortest Paths: Practical Improvements

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

- $d(v)$ = cost of a cheapest $v \to t$ path we have found so far.
- $successor(v)$ = next node on a $v \to t$ 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: Efficient Implementation

Bellman-Ford (V, E, c, t)

For each node $v \in V$
  $d(v) \leftarrow \infty.$
  $successor(v) \leftarrow null.$
$d(t) \leftarrow 0.$

For $i = 0$ To $n - 1$
  For each node $w \in V$
    If $d(w)$ was updated in previous iteration
      For each edge $(v, w) \in E$
        If $d(v) > d(w) + c_{vw}$
          $d(v) \leftarrow d(w) + c_{vw}.$
          $successor(v) \leftarrow w.$
  If no $d(w)$ value changed in iteration i
    Stop