Chapter 12.6

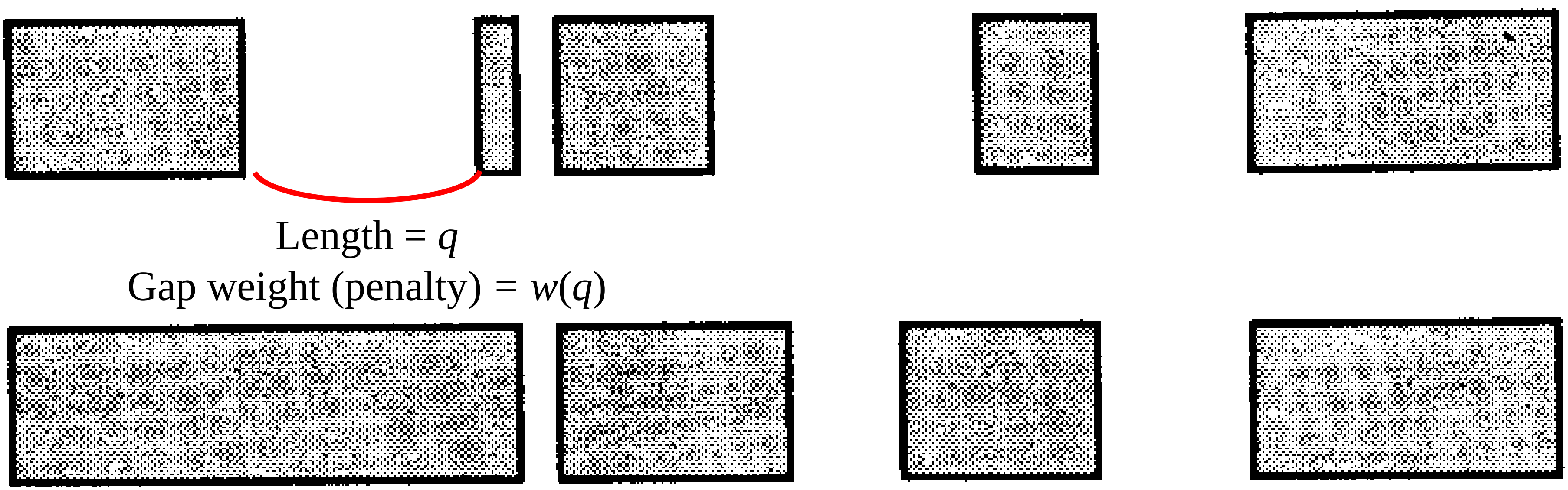
2014. 10. 21

ISA Lab

배준우

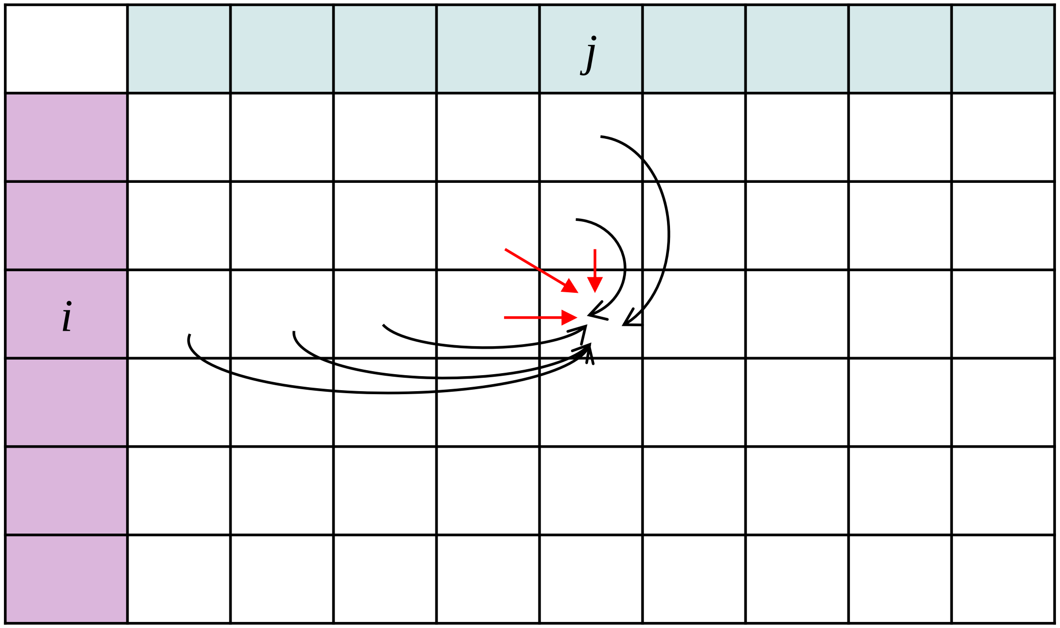
Gap weight

- **Gap weight**



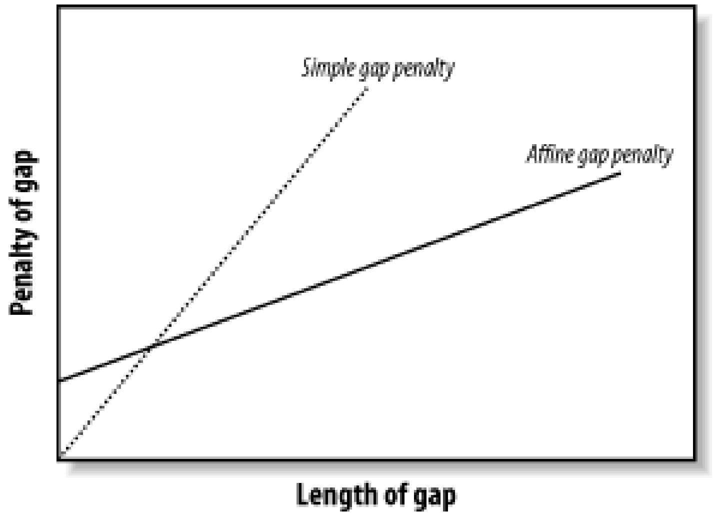
Gap weight

- **Arbitrary gap weight**
  - Any gap weight function is acceptable (this is the most general case)
  - Weight of a gap is an arbitrary function $w(q)$ of its length
  - The constant, affine and convex weight models are of course subcases of the arbitrary weight model



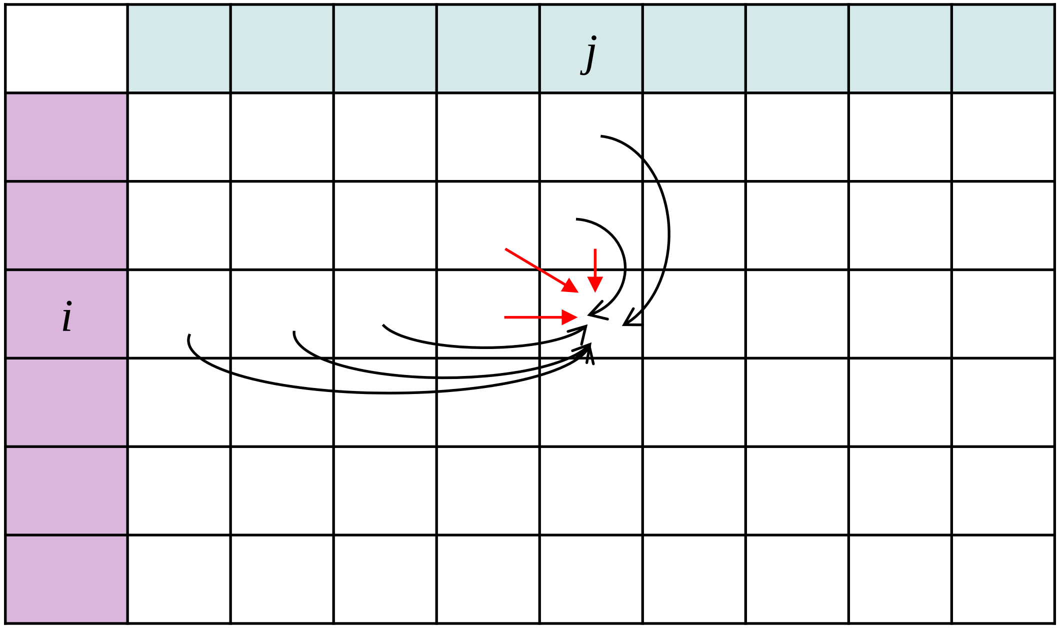
Gap weight

- **Affine gap weight**
  - The model most commonly used by molecular biologists



Gap weight

- **Affine gap weight**
  - The model most commonly used by molecular biologists



Gap weight

- **Convex gap weight**
  - More difficult to solve than with affine gap weights
  - Not as difficult as the problem with arbitrary gap weights

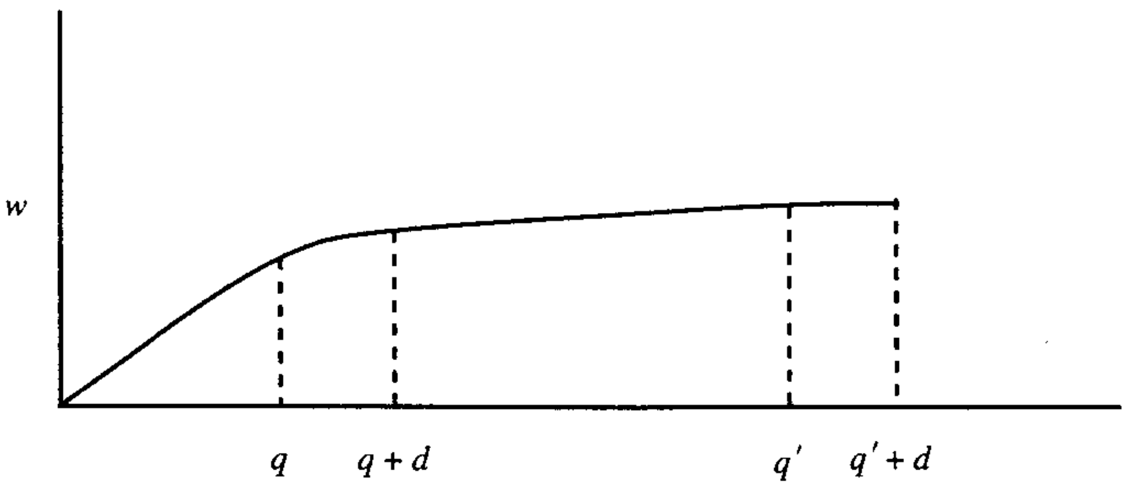


Figure 12.16: A convex function w.

Gap weight

- **Time complexity**
  - Arbitrary gap weights: $O(n^2m)$
  - Affine gap weights: $O(nm)$
  - Convex gap weights: $O(nm\log n)$

  - Affine < Convex < Arbitrary

  - We will discuss convex gap weights in terms of similarity
    - maximum weighted alignment

Convex gap weight

- **Definition**
  - Assume that $w(q)$ is a nonnegative function of length q.
  - Then $w(q)$ is ***convex*** if and only if $w(q+1) - w(q) \leq w(q) - w(q-1)$ for every q.

  - It follows that $w(q+d) - w(q) \geq w(q'+d) - w(q')$ for $q < q'$ for any fixed d.

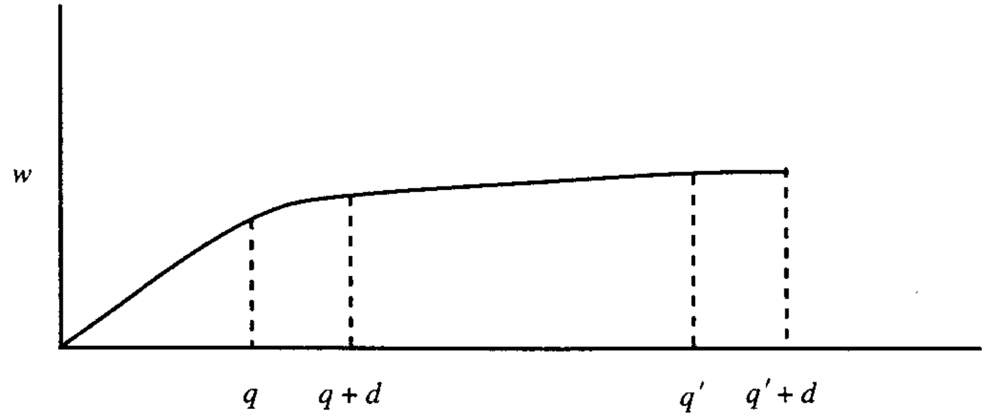
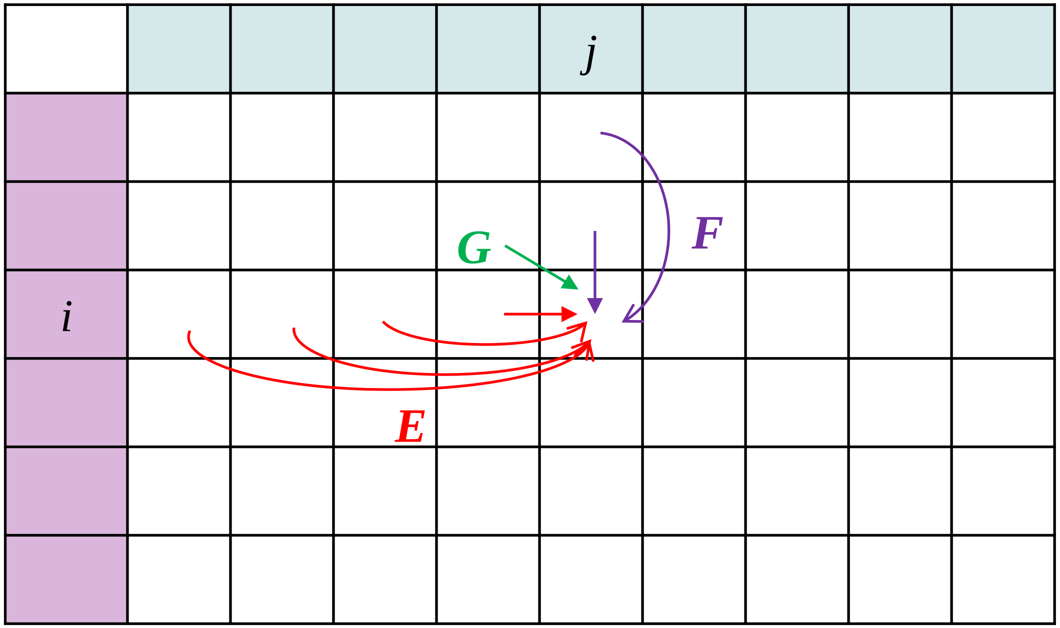


Figure 12.16: A convex function w.

Convex gap weight

- **Speeding up the general recurrences**
  - use the same dynamic programming recurrences developed for arbitrary gap weights
  - but reduce the time needed to evaluate those recurrences



Convex gap weight

- **Speeding up the general recurrences**
  - use the same dynamic programming recurrences developed for arbitrary gap weights
  - but reduce the time needed to evaluate those recurrences

$$\left.\begin{aligned} V(i, j) &= \max[E(i, j), F(i, j), G(i, j)], \\ G(i, j) &= V(i-1, j-1) + s(S_1(i), S_2(j)), \end{aligned}\right\} O(n) \text{ per row}$$

$$\left.\begin{aligned} E(i, j) &= \max_{0 \le k \le j-1} [V(i, k) - w(j-k)], \\ F(i, j) &= \max_{0 \le l \le i-1} [V(l, j) - w(i-l)], \end{aligned}\right\} \begin{aligned} &O(n^2) \text{ per row} \\ &O(m^2) \text{ per column} \end{aligned}$$

$$V(i, 0) = -w(i),$$

$$V(0, j) = -w(j),$$

$$E(i, 0) = -w(i),$$

$$F(0, j) = -w(j).$$

Convex gap weight

- **Simplifying notation**

$$E(i, j) = \max_{0 \le k \le j-1} [V(i, k) - w(j - k)]$$

In any fixed row, we can drop the row index *i*

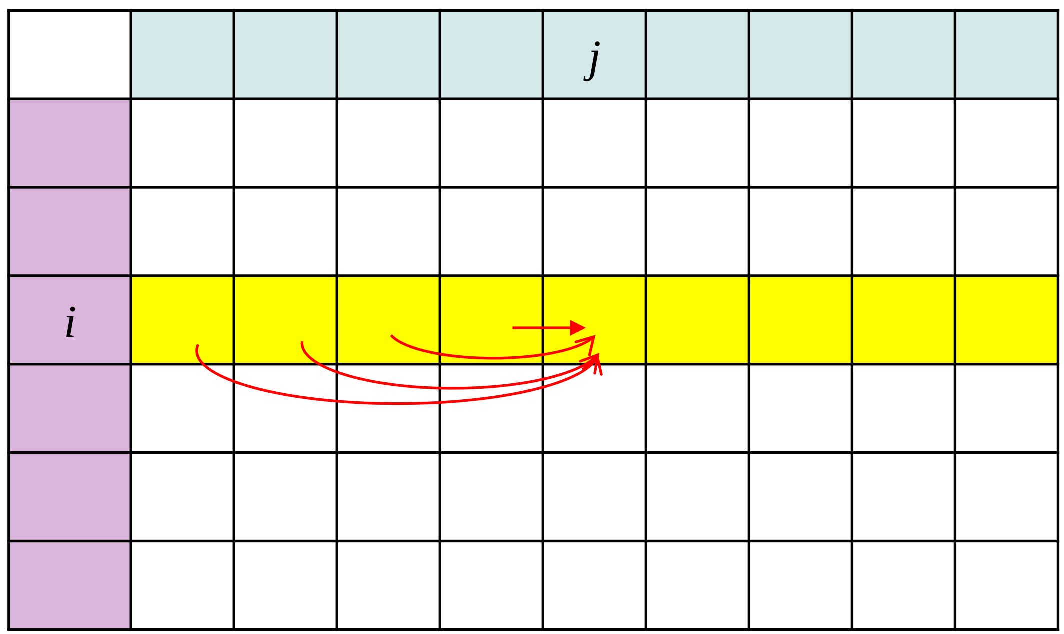
$$E(j) = \max_{0 \le k \le j-1} [V(k) - w(j - k)]$$

$$Cand(k, j) = V(k) - w(j - k)$$

$$E(j) = \max_{0 \le k \le j-1} Cand(k, j)$$

Convex gap weight

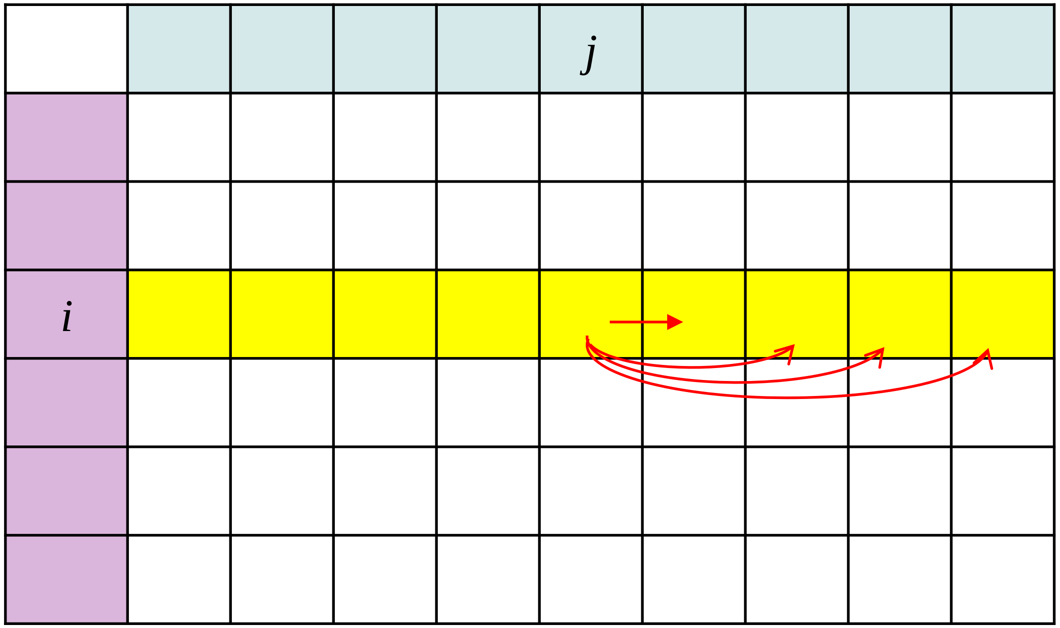
- **Forward dynamic programming**
  - It is more helpful in this exposition



Backward dynamic programming

Convex gap weight

- **Forward dynamic programming**
  - It is more helpful in this exposition



Forward dynamic programming

Convex gap weight

- **Forward dynamic programming**

$\bar{E}(j)$

1	2	…	j-1	j	j+1	…	n-1	n

$E(j)$

1	2	…	j-1	j	j+1	…	n-1	n

Convex gap weight

- **Forward dynamic programming**

$\bar{E}(j)$

1	2	…	j-1	j	j+1	…	n-1	n
$Cand(0,1)$	$Cand(0,2)$	…	$Cand(0,j\text{-}1)$	$Cand(0,j)$	$Cand(0,j+1)$	…	$Cand(0,n\text{-}1)$	$Cand(0,n)$

$E(j)$

1	2	…	j-1	j	j+1	…	n-1	n

Convex gap weight

- **Forward dynamic programming**

$\bar{E}(j)$

1	2	…	j-1	j	j+1	…	n-1	n
$Cand(0,1)$	$Cand(0,2)$	…	$Cand(0,j\text{-}1)$	$Cand(0,j)$	$Cand(0,j+1)$	…	$Cand(0,n\text{-}1)$	$Cand(0,n)$

$E(j)$

1	2	…	j-1	j	j+1	…	n-1	n
$\bar{E}(1)$								

Convex gap weight

- **Forward dynamic programming**

$\bar{E}(j)$

1	2	…	j-1	j	j+1	…	n-1	n
Cand(0,1)	*Cand*(0,2)	…	*Cand*(0,j-1)	*Cand*(0,j)	*Cand*(0,j+1)	…	*Cand*(0,n-1)	*Cand*(0,n)
	Cand(1,2)	…	*Cand*(1,j-1)	*Cand*(1,j)	*Cand*(1,j+1)	…	*Cand*(1,n-1)	*Cand*(1,n)

Assume >

$E(j)$

1	2	…	j-1	j	j+1	…	n-1	n
$\bar{E}(1)$								

Convex gap weight

- **Forward dynamic programming**

$\bar{E}(j)$

1	2	…	j-1	j	j+1	…	n-1	n
$Cand(0,1)$	$Cand(1,2)$	…	$Cand(1,j\text{-}1)$	$Cand(1,j)$	$Cand(1,j+1)$	…	$Cand(1,n\text{-}1)$	$Cand(1,n)$

$Cand(1,2)$	…	$Cand(1,j\text{-}1)$	$Cand(1,j)$	$Cand(1,j+1)$	…	$Cand(1,n\text{-}1)$	$Cand(1,n)$

$E(j)$

1	2	…	j-1	j	j+1	…	n-1	n
$\bar{E}(1)$								

Convex gap weight

- **Forward dynamic programming**

$\bar{E}(j)$

1	2	…	j-1	j	j+1	…	n-1	n
$Cand(0,1)$	$Cand(1,2)$	…	$Cand(1,j\text{-}1)$	$Cand(1,j)$	$Cand(1,j+1)$	…	$Cand(1,n\text{-}1)$	$Cand(1,n)$
		…	$Cand(2,j\text{-}1)$	$Cand(2,j)$	$Cand(2,j+1)$	…	$Cand(2,n\text{-}1)$	$Cand(2,n)$

$E(j)$

1	2	…	j-1	j	j+1	…	n-1	n
$\bar{E}(1)$	$\bar{E}(2)$							

Convex gap weight

Forward dynamic programming for a fixed row

For $j := 1$ to m do
begin
$\bar{E}(j) := Cand(0, j)$;
end;

initialize

For $j := 1$ to m do
begin
$E(j) := \bar{E}(j)$;
$V(j) := \max[G(j), E(j), F(j)]$;
{We assume, but do not show that $F(j)$ and $G(j)$ have been computed for cell j in the row.}

Set $E(j)$ to the current $\bar{E}(j)$

For $j' := j + 1$ to m do {Loop 1}
    if $\bar{E}(j') < Cand(j, j')$ then
        begin
        $\bar{E}(j') := Cand(j, j')$;
        end
end;

Traverse forwards to set $\bar{E}(j')$

Convex gap weight

- **Forward dynamic programming**
  - An alternative way to think about forward dynamic programming
    ->Weighted edit graph for alignment problem

  - Optimal distances = optimal alignments
  - Distance algorithms(such as Dijkstra's algorithm) for shortest distance can be described as forward looking

Convex gap weight

- **Forward dynamic programming**
  - Time complexity
    - Both backward and forward dynamic programming
    - Exactly the same arithmetic operations and comparisons are done
    - Still requires $\Theta(n^2)$ time per row
    - No faster than backwards dynamic programming

Convex gap weight

- **The basis of the speedup**

$\bar{E}(j)$

1	2	…	j-1	j	j+1	…	n-1	n
Cand(0,1)	*Cand*(1,2)	…	*Cand*(5,j-1)	*Cand*(j-1,j)	*Cand*(j-1,j+1)	…	*Cand*(j-1,n-1)	*Cand*(j-1,n)

					Cand(j,j+1)	…	*Cand*(j,n-1)	*Cand*(j,n)

If > ~> ***Win***
else ~> ***lose***

$E(j)$

1	2	…	j-1	j	j+1	…	n-1	n
$\bar{E}(1)$	$\bar{E}(2)$	…	$\bar{E}(j\text{-}1)$	$\bar{E}(j)$				

Convex gap weight

- **The basis of the speedup**
  - *Candidate list* approach
    - The speedup works
      - by identifying and eliminating large numbers of candidate values
      - that have **no chance of winning any comparison**
    - Key observation
      - Let j be the current cell.
      - If $Cand(j, j') \leq \bar{E}(j')$ for some $j' > j$,
      - then $Cand(j,j'') \leq \bar{E}(j'')$ for every $j'' > j'$

Convex gap weight

- **The basis of the speedup**

$\bar{E}(j)$

1	…	j-1	j	j+1	j+2	…	n-1	n
$\bar{E}(1)$	…	$\bar{E}(j\text{-}1)$	$\bar{E}(j)$	$\bar{E}(j+1)$	$\bar{E}(j+2)$	…	$\bar{E}(n\text{-}1)$	$\bar{E}(n)$

$Cand(j,j+1)$	$Cand(j,j+2)$	…	$Cand(j,n\text{-}1)$	$Cand(j,n)$

Convex gap weight

- **The basis of the speedup**

$\bar{E}(j)$

1	…	j-1	j	j+1	j+2	…	n-1	n
$\bar{E}(1)$	…	$\bar{E}(j\text{-}1)$	$\bar{E}(j)$	$Cand(j,j+1)$	$\bar{E}(j+2)$	…	$\bar{E}(n\text{-}1)$	$\bar{E}(n)$

$Cand(j,j+1)$	$Cand(j,j+2)$	…	$Cand(j,n\text{-}1)$	$Cand(j,n)$

Convex gap weight

- **The basis of the speedup**
  - **Lemma 12.6.1**
    - Let $k < j < j' < j''$ be any four cells in the same row.
    - If $Cand(j, j') \leq Cand(k, j')$ then $Cand(j, j'') \leq Cand(k, j'')$

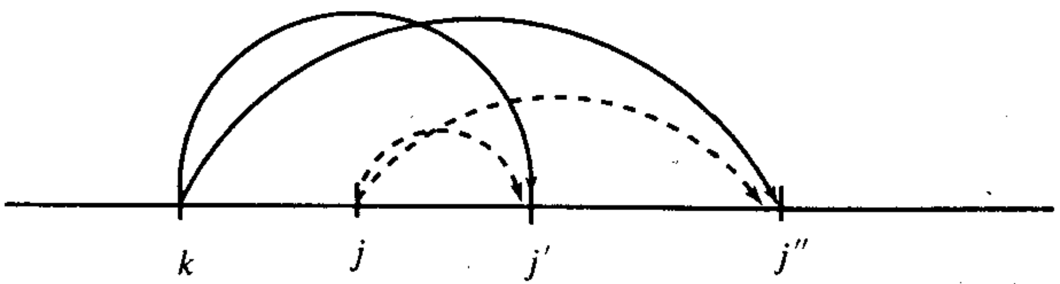


Figure 12.17: Graphical illustration of the *key observation*. Winning candidates are shown with a solid curve and losers with a dashed curve. If the candidate from *j* loses to the candidate from *k* at cell *j'*, then the candidate from *j* will lose to the candidate from *k* at every cell *j''* to the right of *j'*.

Convex gap weight

- **The basis of the speedup**
  - **Lemma 12.6.1**
    - Let $k < j < j' < j''$ be any four cells in the same row.
    - If $Cand(j, j') \leq Cand(k, j')$ then $Cand(j, j'') \leq Cand(k, j'')$
  - PROOF
    - $Cand(k, j) = V(k) - w(j - k)$

$$V(k) - w(j' - k) \geq V(j) - w(j' - j)$$
$$V(k) - V(j) \geq w(j' - k) - w(j' - j)$$

Convex gap weight

- **The basis of the speedup**
  - **Lemma 12.6.1**
    - Let $k < j < j' < j''$ be any four cells in the same row.
    - If $Cand(j, j') \leq Cand(k, j')$ then $Cand(j, j'') \leq Cand(k, j'')$
  - PROOF
    - $Cand(k, j) = V(k) - w(j - k)$

$$V(k) - w(j' - k) \geq V(j) - w(j' - j)$$
$$V(k) - V(j) \geq \mathbf{w(j' - k) - w(j' - j)}$$

    - By convexity, $\mathbf{w(j' - k) - w(j' - j)} \geq \mathbf{w(j'' - k) - w(j'' - j)}$
    - $V(k) - V(j) \geq \mathbf{w(j'' - k) - w(j'' - j)}$
    - $V(k) - \mathbf{w(j'' - k)} \geq V(j) - \mathbf{w(j'' - j)}$

Convex gap weight

- **The basis of the speedup**
  - **Lemma 12.6.1**
    - Let $k < j < j' < j''$ be any four cells in the same row.
    - If $Cand(j, j') \leq Cand(k, j')$ then $Cand(j, j'') \leq Cand(k, j'')$
  - PROOF
    - $Cand(k, j) = V(k) - w(j - k)$

$$V(k) - w(j' - k) \geq V(j) - w(j' - j)$$
$$V(k) - V(j) \geq \mathbf{w(j' - k) - w(j' - j)}$$

    - By convexity, $\mathbf{w(j' - k) - w(j' - j)} \geq \mathbf{w(j'' - k) - w(j'' - j)}$
    - $V(k) - V(j) \geq \mathbf{w(j'' - k) - w(j'' - j)}$
    - $V(k) - \mathbf{w(j'' - k)} \geq V(j) - \mathbf{w(j'' - j)}$

Convex gap weight

- **The basis of the speedup**

Forward dynamic programming for a fixed row

For $j := 1$ to m do
begin
$\overline{E}(j) := Cand(0, j);$
end;

For $j := 1$ to m do
begin
$E(j) := \overline{E}(j);$
$V(j) := \max[G(j), E(j), F(j)];$
{We assume, but do not show that $F(j)$ and $G(j)$ have been computed for cell j in the row.}

For $j' := j + 1$ to m do {Loop 1}
    if $\overline{E}(j') < Cand(j, j')$ then
        begin
        $\overline{E}(j') := Cand(j, j');$
        end
end;

But this improvement does not lead directly to a better (worst-case) time bound

else then
    break

Convex gap weight

- **Cell pointers and row partition**

Forward dynamic programming for a fixed row

```
For j := 1 to m do
begin
Ē(j) := Cand(0, j);
b(j) := 0
end;

For j := 1 to m do
begin
E(j) := Ē(j);
V(j) := max[G(j), E(j), F(j)];
{We assume, but do not show that F(j) and G(j)
have been computed for cell j in the row.}

For j' := j + 1 to m do {Loop 1}
      if Ē(j') < Cand(j, j') then
            begin
            Ē(j') := Cand(j, j');
            b(j') := j; {This sets a pointer from j' to j to be explained later.}
            end
end;
```

Convex gap weight

- **Cell pointers and row partition**

$\bar{E}(j)$

1	2	…	j-1	j	j+1	…	n-1	n
$\bar{E}(1)$	$\bar{E}(2)$	…	*Cand*(2,j-1)	*Cand*(2,j)	*Cand*(1,j+1)	…	*Cand*(1,n-1)	*Cand*(1,n)

Convex gap weight

- **Cell pointers and row partition**
  - Lemma 12.6.2
    - Consider the point when j is the current cell
    - but before j sends forward any candidate values
    - At that point, $b(j') \geq b(j'+1)$ for every cell j' from $j+1$ to m-1

$\bar{E}(j)$

1	…	j-1	j	j+1	j+2	…	n-1	n
$\bar{E}(1)$	…	$\bar{E}(j\text{-}1)$	$\bar{E}(j)$	***Cand*(*j*-1,*j*+1)**	***Cand*(*j*,*j*+2)**	…	*Cand*(1,n-1)	*Cand*(1,n)

Convex gap weight

- **Cell pointers and row partition**
  - Lemma 12.6.2
    - Consider the point when j is the current cell
    - but before j sends forward any candidate values
    - At that point, $b(j') \geq b(j'+1)$ for every cell j' from $j+1$ to $m-1$
  - PROOF
    - Suppose $b(j') < b(j'+1)$
    - By Lemma 12.6.1, $Cand(b(j'), j'+1) \geq Cand(b(j'+1), j'+1)$
    - $b(j'+1)$ should be set to $b(j')$, not $b(j'+1)$
    - Hence $b(j') \geq b(j'+1)$

Convex gap weight

- **Cell pointers and row partition**
  - Corollary 12.6.1
    - j is the current cell, but before j sends forward any candidates
    - The values of the b pointers form a nonincreasing sequence from left to right
    - Therefore, cells j, j+1, j+2, … , m are partitioned into maximal blocks of consecutive cells
      - such that all b pointers in the block have the same value
      - and the pointer values decline in successive blocks

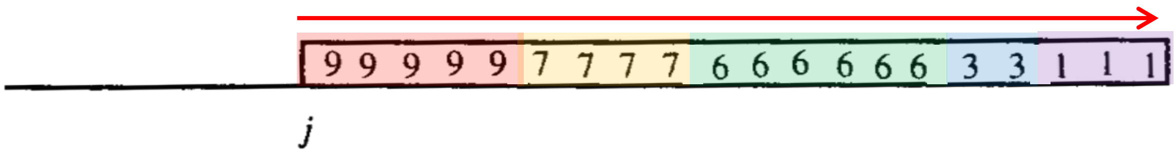


Figure 12.18: Partition of the cells $j+1$ through m into maximal blocks of consecutive cells such that all the cells in any block have the same b value. The common b value in any block is less than the common b value in the preceding block.

Convex gap weight

- **Cell pointers and row partition**
  - Given Corollary 12.6.1
    - The algorithm doesn't need to explicitly maintain a *b* pointer for every cell
    - Only record the **common *b* pointer** for each block

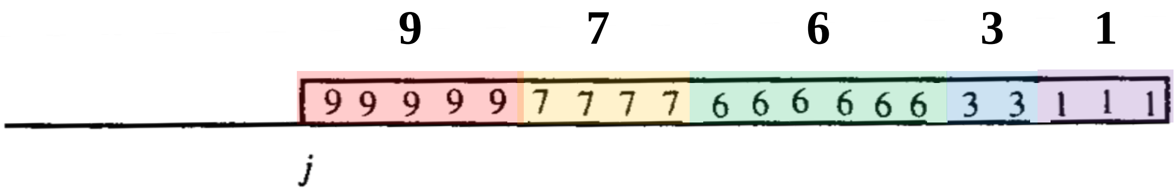


Figure 12.18: Partition of the cells $j+1$ through m into maximal blocks of consecutive cells such that all the cells in any block have the same b value. The common b value in any block is less than the common b value in the preceding block.

Convex gap weight

- **Preparation for the speedup**
  - Our goal
    - To reduce the time per row used in computing the E values
    - $\Theta(n^2)$ -> $O(n \log n)$
  - The main work done in a row

    1. Update the $\bar{E}$ values

    2. Update the current block-partition with its associated pointer

Convex gap weight

- **Preparation for the speedup**
  - Our goal
    - To reduce the time per row used in computing the E values
    - $\Theta(n^2)$ -> $O(n \log n)$
  - The main work done in a row

    1. Update the $\bar{E}$ values
       - Assume that all the $\bar{E}$ values are maintained for free

    2. **Update the current block-partition** with its associated pointer

Convex gap weight

- **Preparation for the speedup**

$\bar{E}(j)$

1	2	…	j-1	j	j+1	…	n-1	n
$Cand(0,1)$	$Cand(0,2)$	…	$Cand(0,j-1)$	$Cand(0,j)$	$Cand(0,j+1)$	…	$Cand(0,n-1)$	$Cand(0,n)$

$E(j)$

1	2	…	j-1	j	j+1	…	n-1	n

Convex gap weight

- **Preparation for the speedup**

$b = 0$

$\bar{E}(j)$

1	2	…	j-1	j	j+1	…	n-1	n
$Cand(0,1)$	$Cand(0,2)$	…	$Cand(0,j-1)$	$Cand(0,j)$	$Cand(0,j+1)$	…	$Cand(0,n-1)$	$Cand(0,n)$

$E(j)$

1	2	…	j-1	j	j+1	…	n-1	n
$\bar{E}(1)$								

Convex gap weight

- **Preparation for the speedup**

There are only **three cases** for the new block-partition

$b = 0$

$\bar{E}(j)$

1	2	…	j-1	j	j+1	…	n-1	n
$Cand(0,1)$	$Cand(0,2)$	…	$Cand(0,j\text{-}1)$	$Cand(0,j)$	$Cand(0,j+1)$	…	$Cand(0,n\text{-}1)$	$Cand(0,n)$

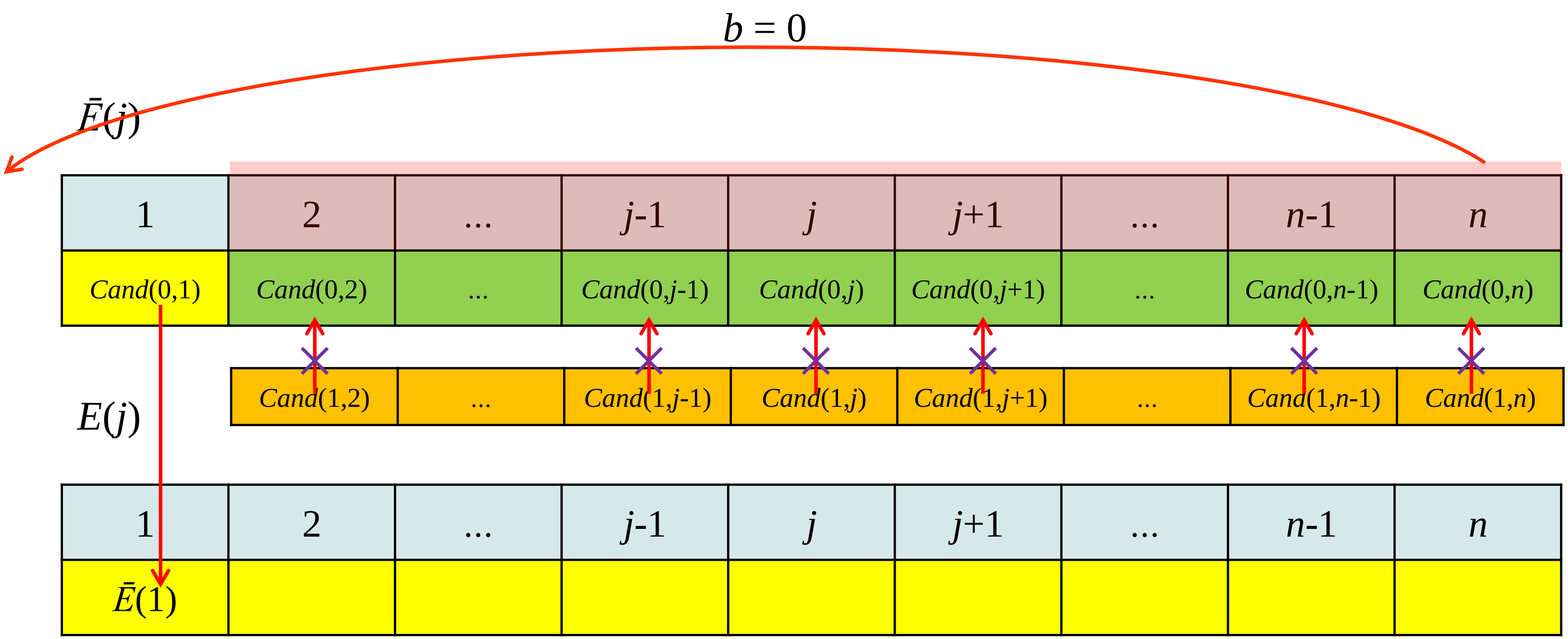
$E(j)$

$Cand(1,2)$	…	$Cand(1,j\text{-}1)$	$Cand(1,j)$	$Cand(1,j+1)$	…	$Cand(1,n\text{-}1)$	$Cand(1,n)$

1	2	…	j-1	j	j+1	…	n-1	n
$\bar{E}(1)$								

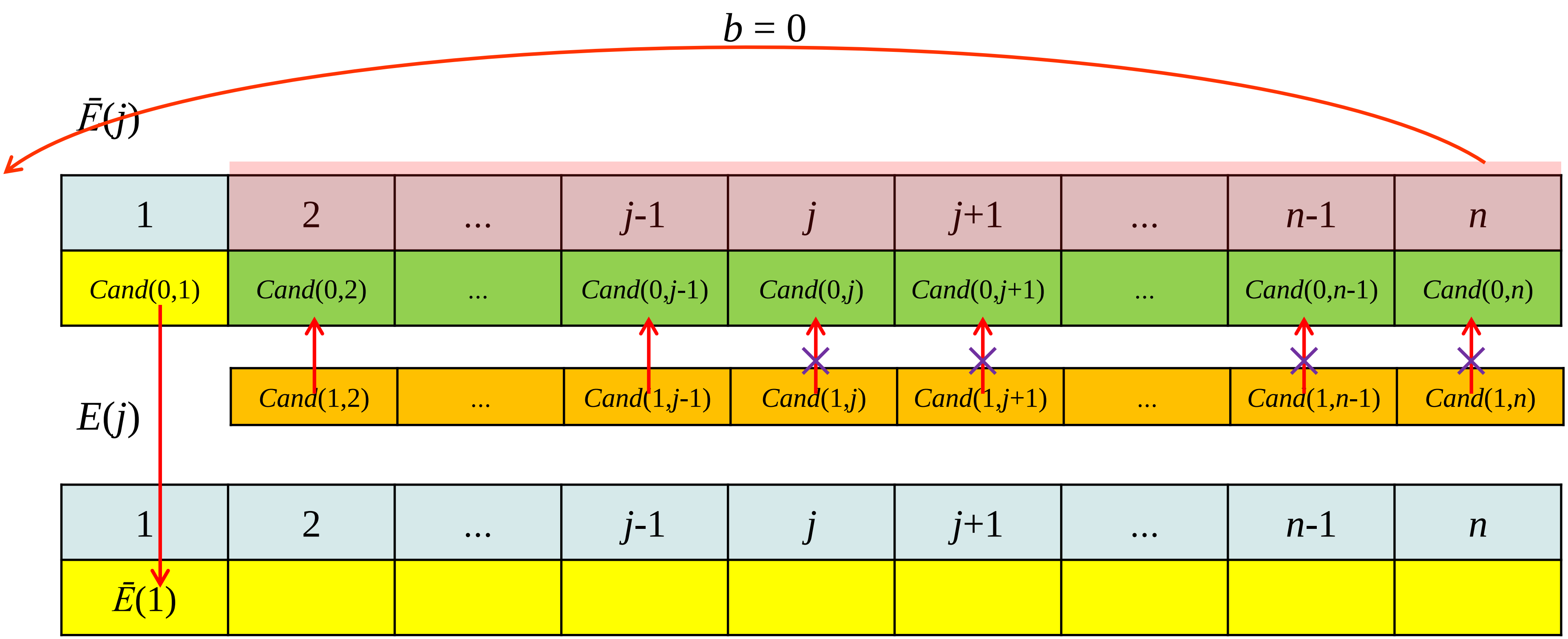
Convex gap weight

- **Preparation for the speedup**
  - Case 1: remain in a **single block** with common pointer ***b* = 0**



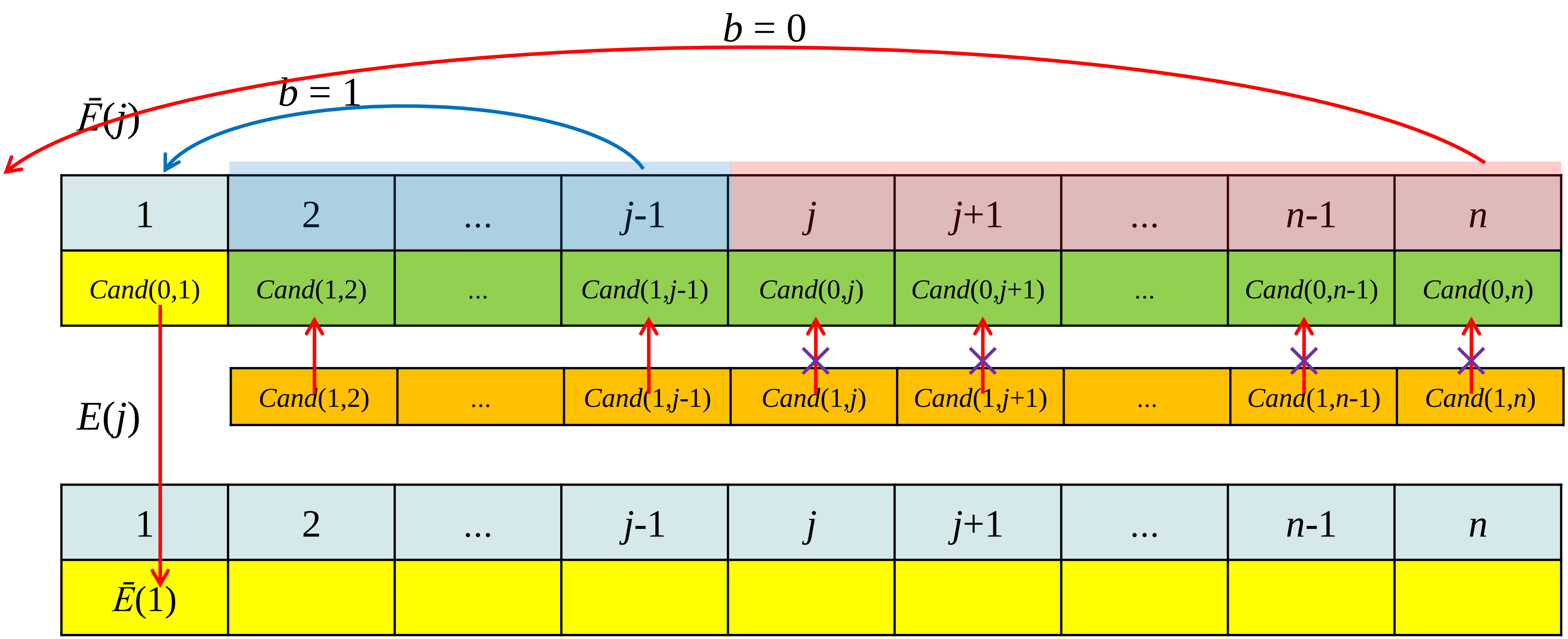
Convex gap weight

- **Preparation for the speedup**
  - Case 2: divided into **two blocks**



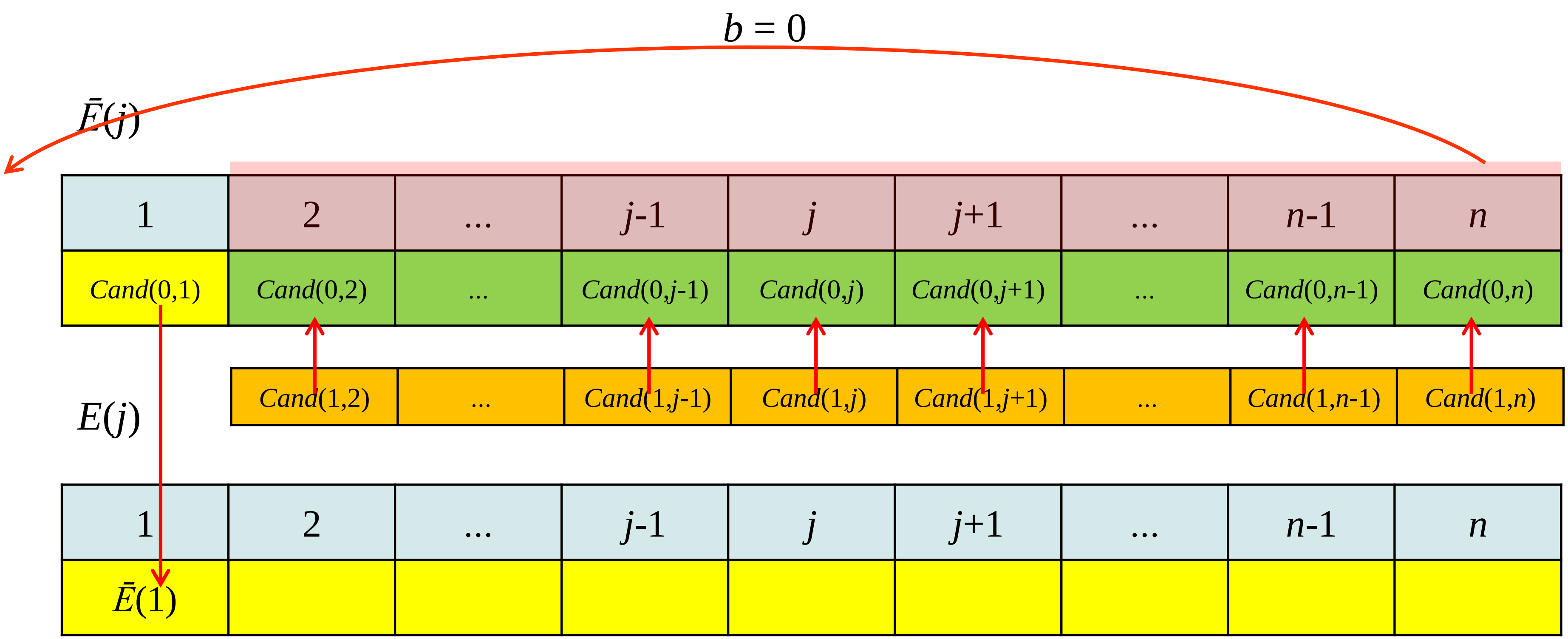
Convex gap weight

- **Preparation for the speedup**
  - Case 2: divided into **two blocks**



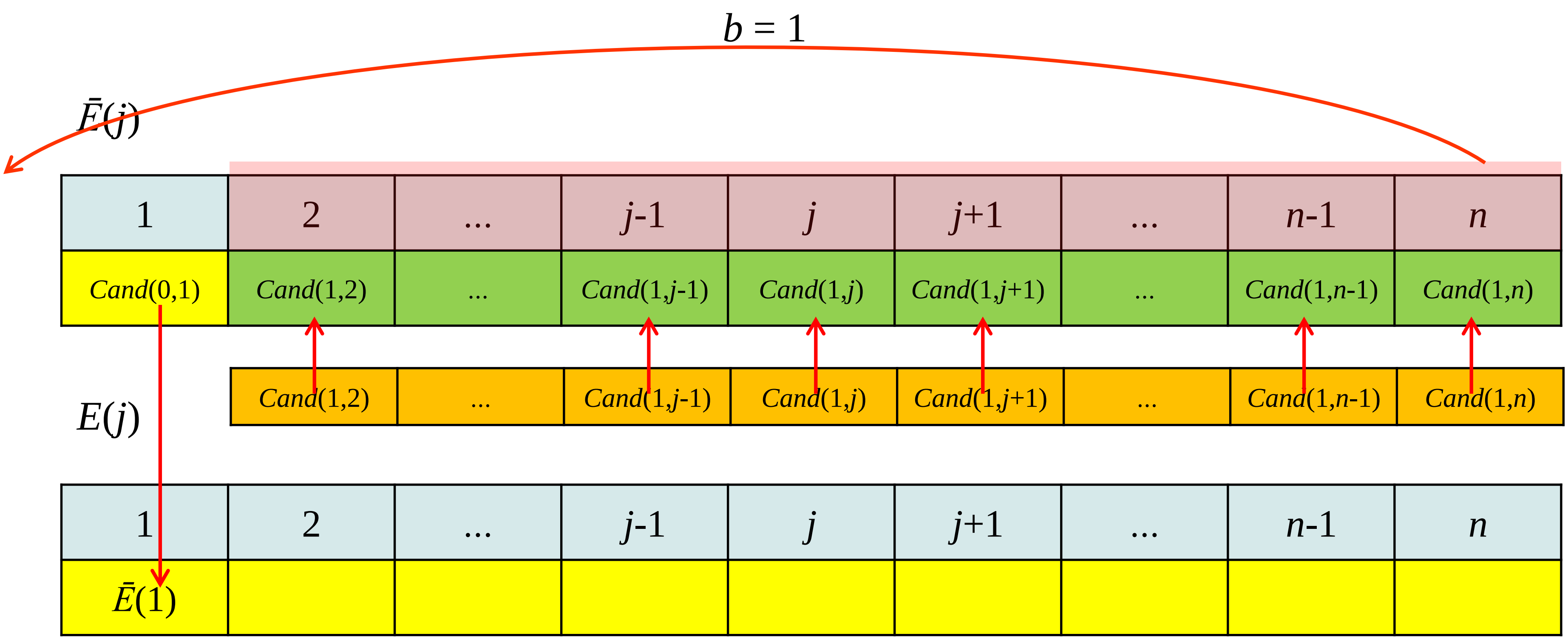
Convex gap weight

- **Preparation for the speedup**
  - Case 3: remain in a **single block** with common pointer ***b* = 1**



Convex gap weight

- **Preparation for the speedup**
  - Case 3: remain in a **single block** with common pointer ***b* = 1**



Convex gap weight

- **Preparation for the speedup**
  - The punch line
    - Search for the left-most cell j' >2 such that $\bar{E}(j') \geq Cand(1, j')$
      - *Win … Win* ***Lose*** *Lose … Lose*
    - can be done by **binary search**
    - $O(\log n)$ comparisons
    - at most one pointer update
      - Since we only record one b pointer per block

Convex gap weight

- **Preparation for the speedup**
  - General case of $j > 1$

r blocks

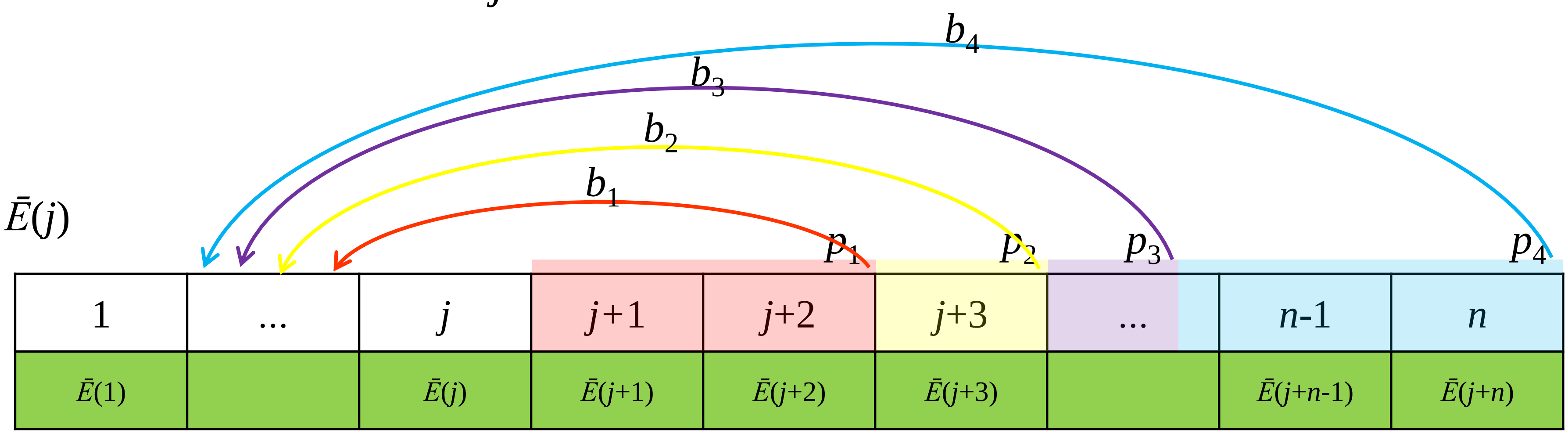
$\bar{E}(j)$

$p_1 = j+2$ $p_2 = j+3$ p_3 $p_4 = p_r = n$

1	...	j	$j+1$	$j+2$	$j+3$	...	n-1	n
$\bar{E}(1)$		$\bar{E}(j)$	$\bar{E}(j+1)$	$\bar{E}(j+2)$	$\bar{E}(j+3)$		$\bar{E}(j+n\text{-}1)$	$\bar{E}(j+n)$

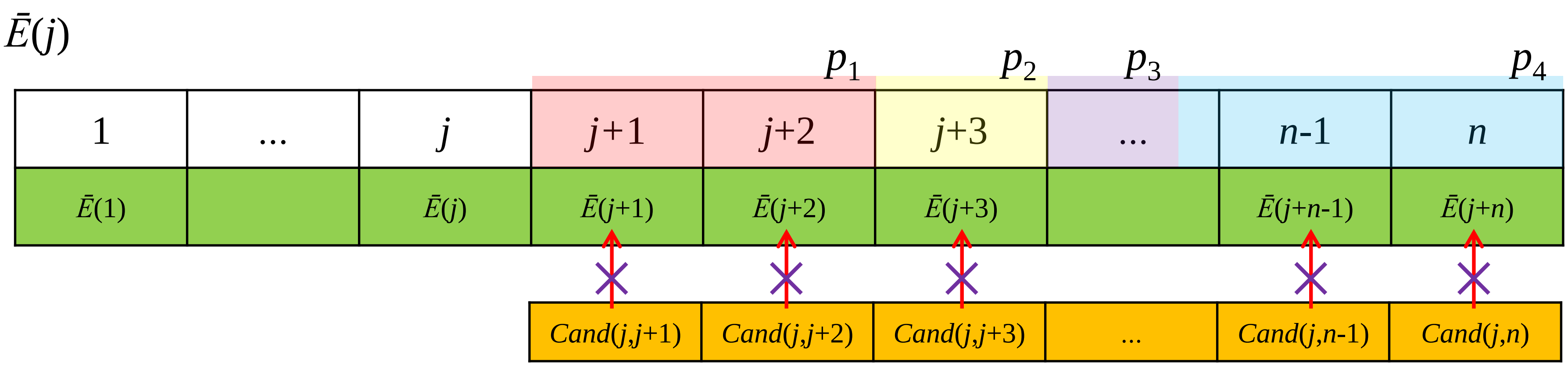
Convex gap weight

- **Preparation for the speedup**
  - General case of $j > 1$



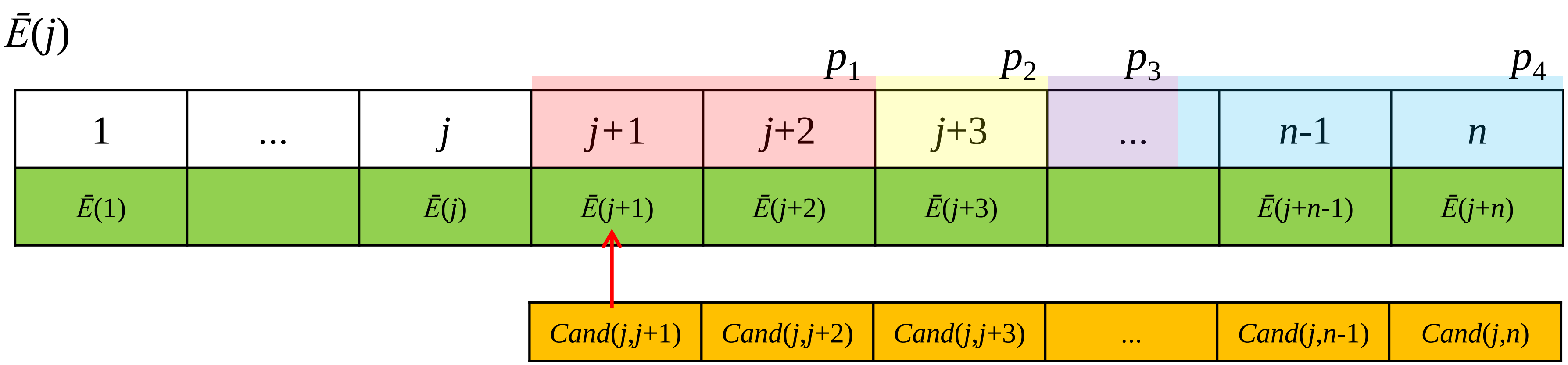
Convex gap weight

- **Preparation for the speedup**
  - Case 1: if $\bar{E}(j+1) \geq Cand\,(j, j+1)$ ***Lose***



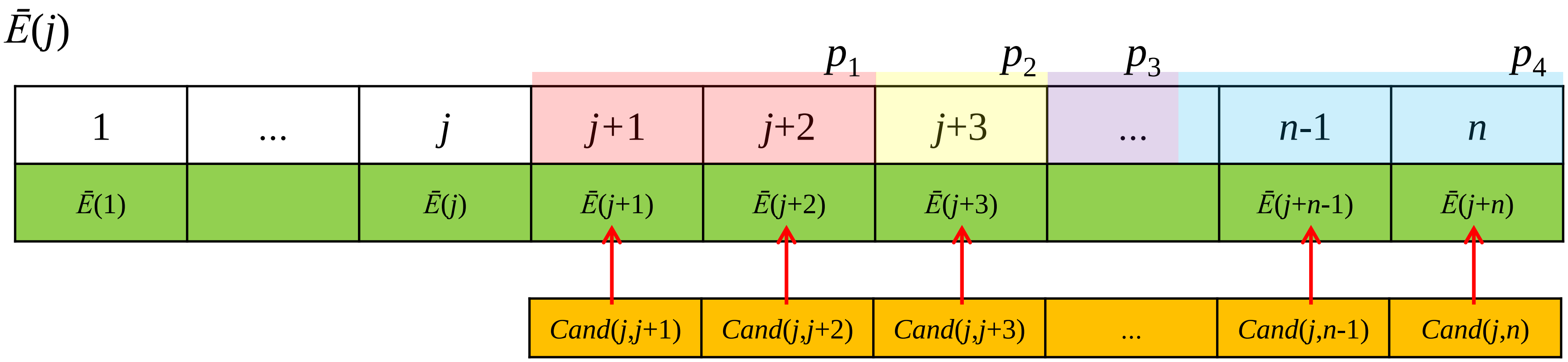
Convex gap weight

- **Preparation for the speedup**
  - Case 2: if $\bar{E}(j+1) < Cand\,(j, j+1)$ ***Win***
    - A. until end-of-block list is exhausted
    - B. until it finds the first index s with $\bar{E}(p_s) \geq Cand\,(j, p_s)$



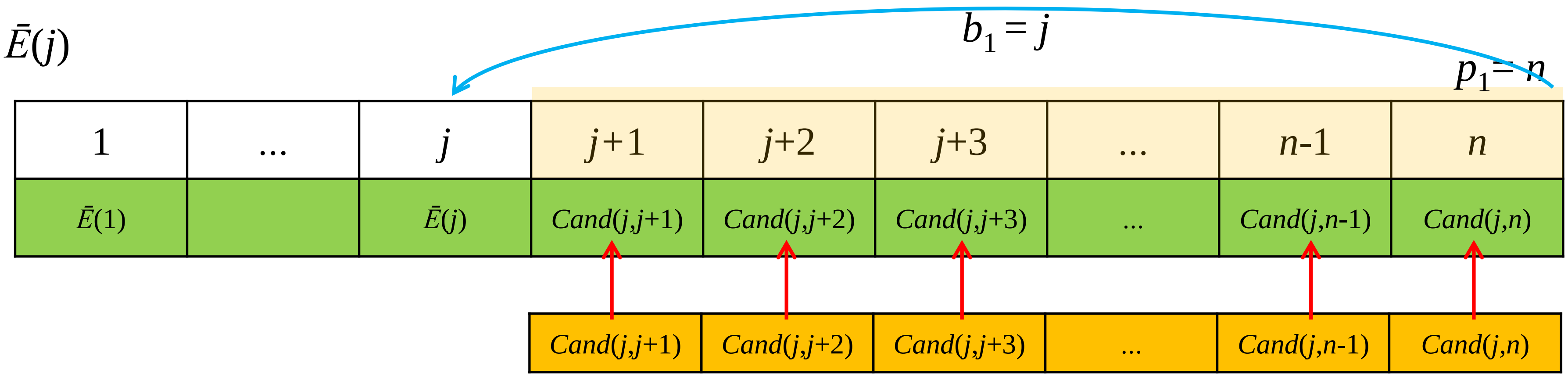
Convex gap weight

- **Preparation for the speedup**
  - Case 2: if $\bar{E}(j+1) < Cand\,(j, j+1)$ ***Win***
    - **A. until end-of-block list is exhausted**
    - B. until it finds the first index s with $\bar{E}(p_s) \geq Cand\,(j, p_s)$



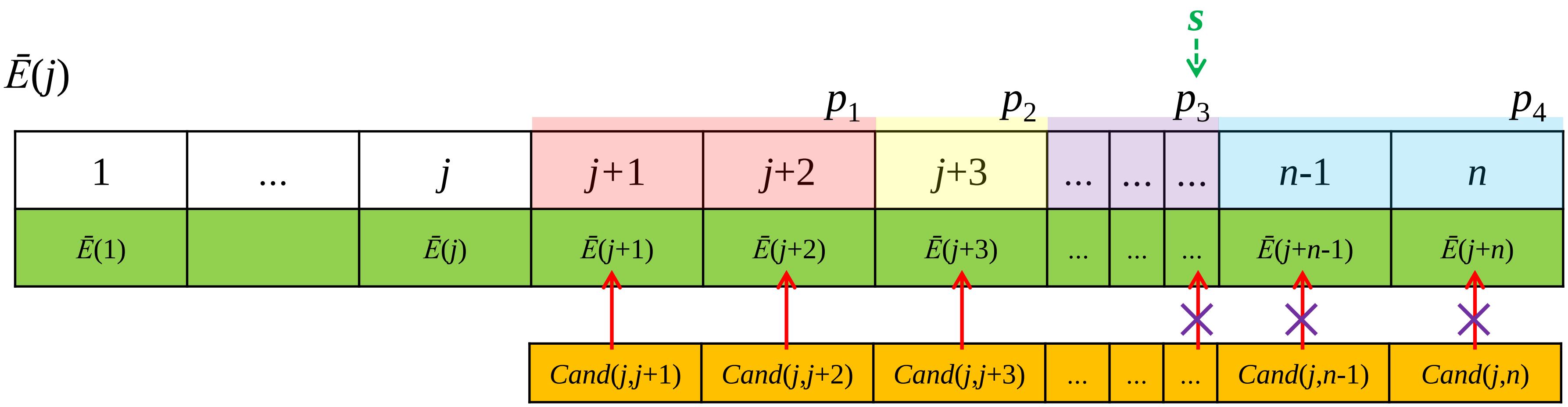
Convex gap weight

- **Preparation for the speedup**
  - Case 2: if $\bar{E}(j+1) < Cand\,(j, j+1)$ ***Win***
    - **A. until end-of-block list is exhausted**
    - B. until it finds the first index s with $\bar{E}(p_s) \geq Cand\,(j, p_s)$



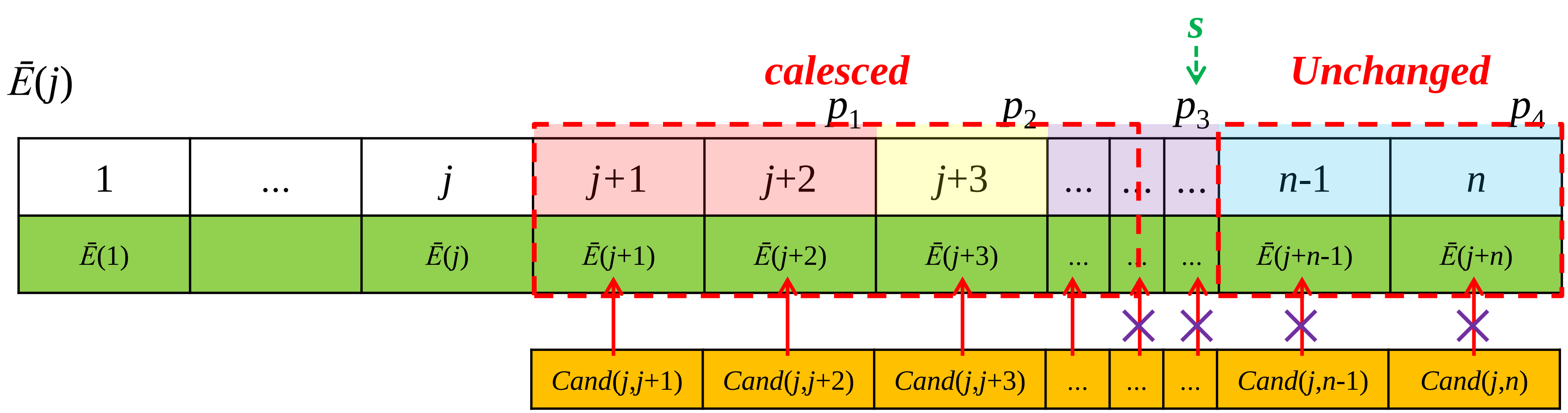
Convex gap weight

- **Preparation for the speedup**
  - Case 2: if $\bar{E}(j+1) < Cand\ (j, j+1)$ ***Win***
    A. until end-of-block list is exhausted
    B. until it finds the first index s with $\bar{E}(p_s) \geq Cand\ (j, p_s)$



Convex gap weight

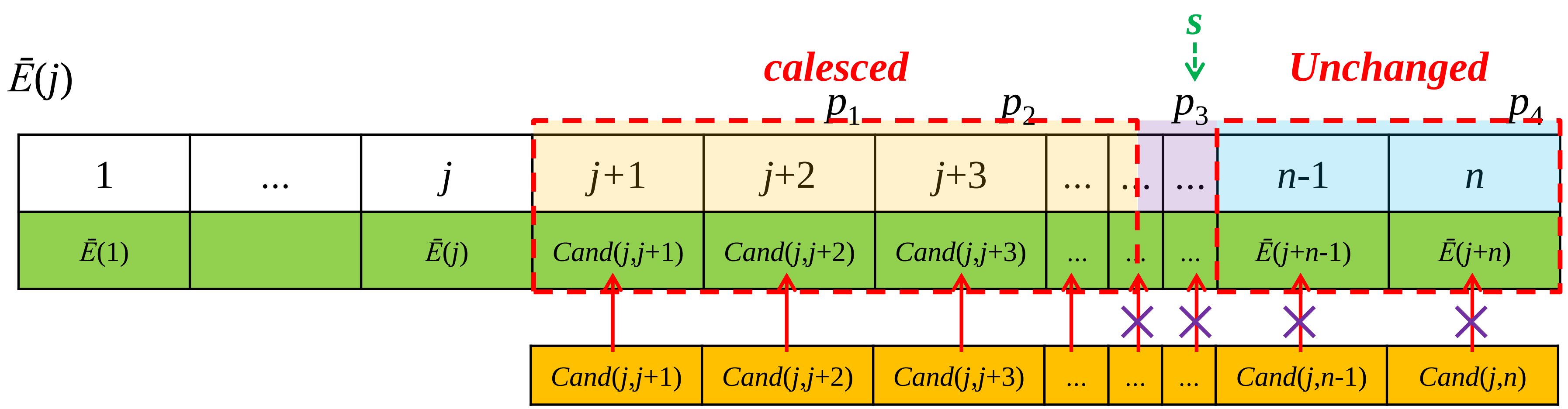
- **Preparation for the speedup**
  - Case 2: if $\bar{E}(j+1) < Cand\,(j, j+1)$ ***Win***
    A. until end-of-block list is exhausted
    B. until it finds the first index s with $\bar{E}(p_s) \geq Cand\,(j, p_s)$



- The algorithm finds the proper place to split block s
  - by doing binary search over the cells in the block

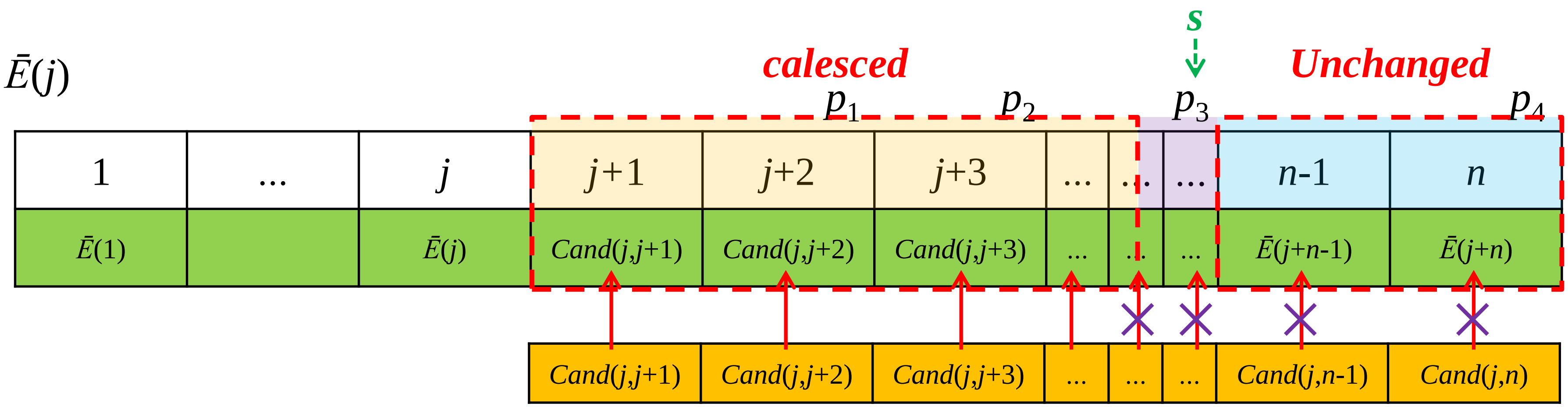
Convex gap weight

- **Preparation for the speedup**
  - Case 2: if $\bar{E}(j+1) < Cand\ (j, j+1)$ ***Win***
    A. until end-of-block list is exhausted
    B. until it finds the first index s with $\bar{E}(p_s) \geq Cand\ (j, p_s)$



Convex gap weight

- **Preparation for the speedup**
  - Case 2: if $\bar{E}(j+1) < Cand\,(j, j+1)$ ***Win***
    A. until end-of-block list is exhausted
    B. until it finds the first index s with $\bar{E}(p_s) \geq Cand\,(j, p_s)$



Convex gap weight

- **Final implementation details**
  - Our goal
    - To reduce the time per row used in computing the E values
    - $\Theta(n^2)$ -> $O(n \log n)$
  - The main work done in a row

    1. **Update the $\bar{E}$ values**
       - Assume that all the $\bar{E}$ values are maintained for free

    2. Update the current block-partition with its associated pointer

Convex gap weight

- **Final implementation details**
  - The key observation
    - retrieves $\bar{E}(j)$
      - only when j is the current cell
    - retrieves $\bar{E}(j')$
      - only when examining cell j' in the process of updating the block-partition

Convex gap weight

- **Final implementation details**
  - The key observation
    - retrieves $\bar{E}(j)$
      - only when j is the current cell
      - j is always in the first block of the current block-partition
        - $b(j) = b_1$
      - $\boldsymbol{\bar{E}(j) = Cand(b_1, j)}$
        - which can be computed in **constant time** when needed
    - retrieves $\bar{E}(j')$
      - only when examining cell j' in the process of updating the block-partition

Convex gap weight

- **Final implementation details**
  - The key observation
    - retrieves $\bar{E}(j)$
      - only when j is the current cell
      - j is always in the first block of the current block-partition
        - $b(j) = b_1$
      - $\boldsymbol{\bar{E}(j) = Cand(b_1, j)}$
        - which can be computed in **constant time** when needed
    - retrieves $\bar{E}(j')$
      - only when examining cell j' in the process of updating the block-partition
      - The algorithm knows the block that j' falls into, say block i,
      - and hence it knows b_i
      - $\boldsymbol{\bar{E}(j') = Cand(b_i, j')}$
        - which can be computed in **constant time** when needed

Convex gap weight

- **Final implementation details**

Revised forward dynamic programming for a fixed row

Initialize the end-of-block list to contain the single number m.
Initialize the associated pointer list to contain the single number 0.

For $j := 1$ to m do
begin
  Set k to be the first pointer on the b-pointer list.
  $E(j) := Cand(k, j)$;
  $V(j) := \max[G(j), E(j), F(j)]$;
  {As before we assume that the needed F and G values have been computed.}

Convex gap weight

- **Final implementation details**

{Now see how j's candidates change the block-partition.}
Set j' equal to the first entry on the end-of-block list.

{look for the first index s in the end-of-block list where j loses}
If $Cand(b(j'), j+1) < Cand(j, j+1)$ then {j's candidate wins one}
begin
While
The end-of-block list is not empty and $Cand(b(j'), j') < Cand(j, j')$ do
begin
remove the first entry on the end-of-block list,
and remove the corresponding b-pointer
If the end-of-block list is not empty then
set j' to the new first entry on the end-of-block list.
end;
end {while};

Convex gap weight

- **Final implementation details**

If the end-of-block list is empty then
place m at the head of that list;
Else {when the end-of-block list is not empty}
begin
Let p_s denote the first end-of-block entry.
Using binary search over the cells in block s, find the
right-most point p in that block such that $Cand(j, p) > Cand(b_s, p)$.
Add p to the head of the end-of-block list;
end;

Add j to the head of the b pointer list.

end;

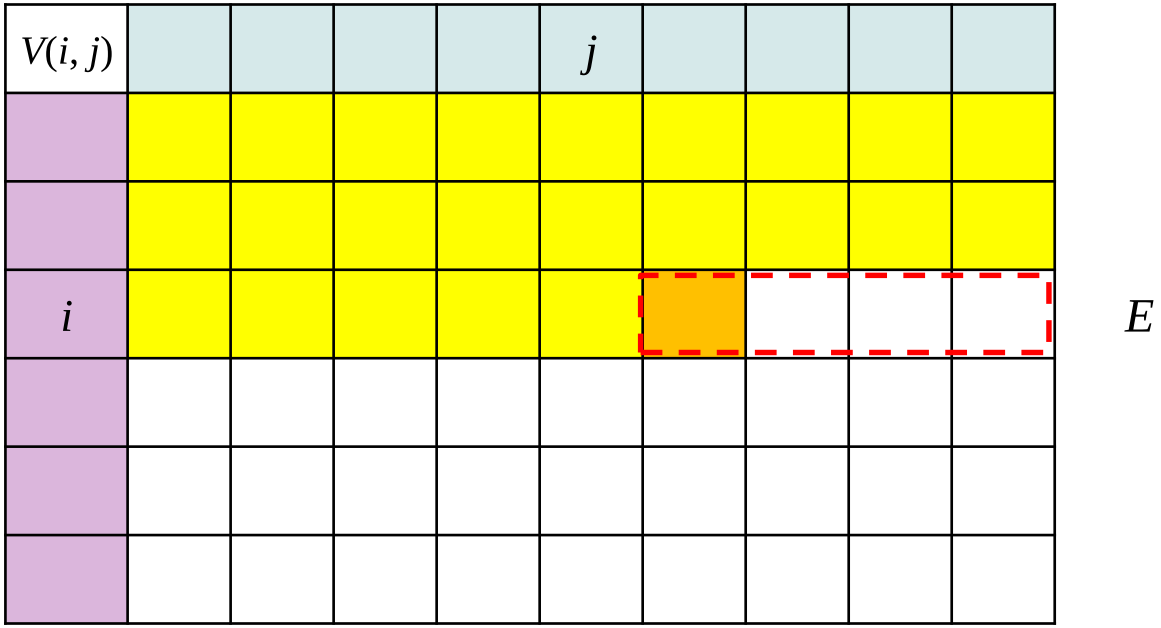
end.

Convex gap weight

- **Time analysis**
  - The total time for the algorithm
    - Proportional to the number of comparisons
      - Find block *s* ~> *l* >2 comparisons, at least *l* -1 full blocks coalesce into a single block
      - Binary search to split *s* ~> splits at most one block into two
    - Hence if the algorithm does ***l* > 2 comparisons** to find *s*,
    - Then the total number of blocks **at most increases by one**.
    - The algorithm begins with a single block and *n* iterations
      - To find every *s*: At most ***O*(*n*)** comparisons
      - To find split cell (binary search): ***O*(log *n*)**
  - **Theorem 12.6.1**
    - For any fixed row, all the *E*(*j*) values can be computed in ***O*(*n* log *n*)** total time.

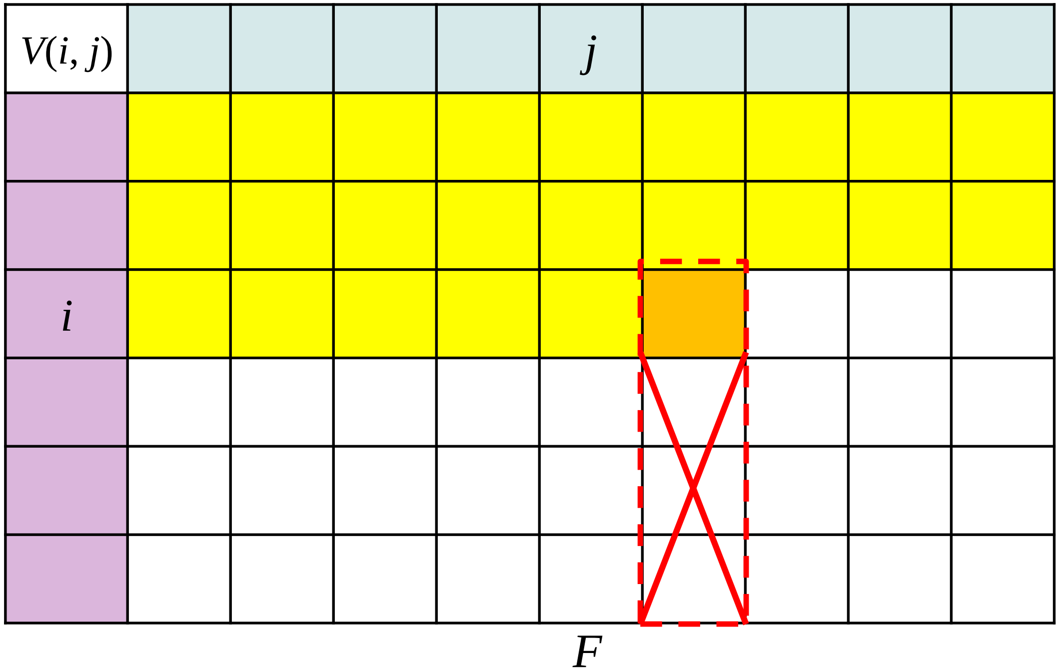
Convex gap weight

- **The case of *F* values is essentially symmetric**
  - One point that might cause confusion
    - The computations of E and F are actually *interleaved*
      - since each $V(i, j)$ value depends on both $E(i, j)$ and $F(i, j)$



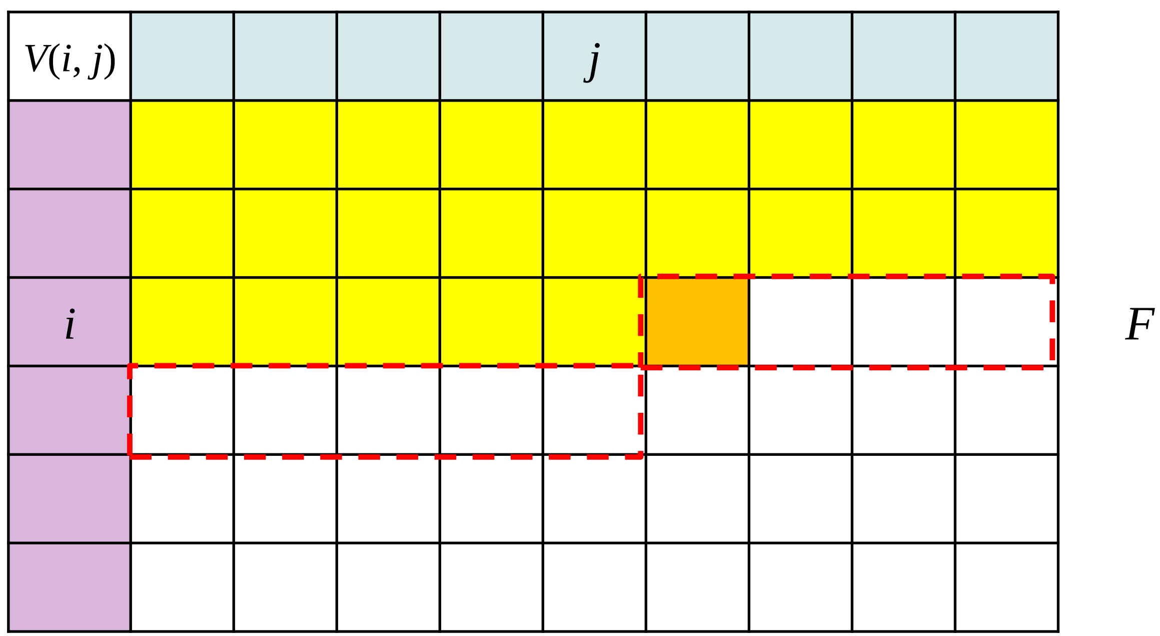
Convex gap weight

- **The case of *F* values is essentially symmetric**
  - One point that might cause confusion
    - The computations of *E* and *F* are actually *interleaved*
      - since each $V(i, j)$ value depends on both $E(i, j)$ and $F(i, j)$



Convex gap weight

- **The case of *F* values is essentially symmetric**
  - One point that might cause confusion
    - The computations of *E* and *F* are actually *interleaved*
      - since each $V(i, j)$ value depends on both $E(i, j)$ and $F(i, j)$



Convex gap weight

- **Total time**
  - E value: $O(n \log n)$
  - F value: $O(m \log m)$
  - G value: $O(nm)$

  - V value: $O(nm)$ once E and F is known

  - **Theorem 12.6.2**
    - When the gap weight w is a convex function of the gap length
    - An optimal alignment can be computed in $\boldsymbol{O(nm \log n)}$ time
      - where $n > m$ are the lengths of the two strings