Why Align Strings?

- Find small differences between strings
  - Differences ~every 100 characters in DNA
- See if the suffix of one sequence is a prefix of another
  - Useful in shotgun sequencing
- Find common subsequences (*cf* definition)
  - Homology or identity searching
- Find similarities of members of the same family
  - Structure prediction

Alignment

- Not an exact match
- Can be based on edit distance
- Usually based on a similarity measure

Metrics

A metric $\rho: X \rightarrow \Re$ is a function with the following properties for $a,b,c \in \Re$

- $P(a) \in \Re$, $\rho \geq 0$ *(real, non-negative)*
- $\rho(a,a)=0$ *(identity)*
- $\rho(a,b)= \rho(b,a)$ *reflexive*
- $\rho(a,c) \leq \rho(a,b)+ \rho(b,c)$ (*triangle inequality)*

Often ρ is called a 'distance'

Edit Distance

The number of changes requires to change one sequence into another is called the ***edit distance.***

V	*I*	*N*	*T*	*N*	*E*	*R*	*S*
W	*I*	*N*	*E*	*Y*	*A*	*R*	*D*

Edit Distance = 5

Similarity

We are more inclined to use the concept of ***similarity***, an alignment scoring function instead. We can then

- deal with gaps
- weight specific substitutions.

Note that similarity is NOT A METRIC.

Example of a Scoring Function for Similarity

Match	+1
Mismatch (replacement)	-1
Align with gap (insertion or deletion) Called “Indels” by Waterman	-2

Similarity Scoring of an Alignment

Example of Two of 6 Possible Alignments

A	T	G	C	A	T	
C	T	$-$	G	C	T	$= -3$
-1	1	-2	-1	-1	1	

A	T	G	C	A	T	
C	T	G	C	$-$	T	$= 1$
-1	1	1	1	-2	1	

String (Sequence) Alignment

- Global Alignment
  - Every character in the query (source) string lines up with a character in the target string
  - May require gap (space) insertion to make strings the same length
- Local Alignment
  - An “internal” alignment or embedding of a substring (*sic*) into a target string

Global *vs* Local

GLOBAL

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

LOCAL

A	*T*	*G*	*A*	*T*	*A*	*C*	*C*	*C*	*T*
	T	*G*	*A*	*A*	*A*	*G*	*G*		

Optimal Global Alignments

A	T	G	C	A	T	
C	T	$-$	G	C	T	$=-3$
-1	1	-2	-1	-1	1	

A	T	G	C	A	T	
C	T	G	C	$-$	T	$=1$
-1	1	1	1	-2	1	

In the earlier example repeated here, the second alignment is obviously better.

How do we know it is optimal?

In this example there are only 6 possible alignments; in a long string the number can become very large.

The Size of the String Alignment Problem

Consider a string of length n to be aligned with another string that has g gaps ($g \leq n/2$)

- With 1 gap there are n places to put the gap
- With 2 gaps there are n-1 places to put the second gap
- With g gaps, there are $n-g+1$ places to put the g^{th} gap or $n(n-1)(n-2)\times\ldots\times(n-g+1)$ possibilities for all gaps

Thus there are precisely $n!/(n-g)!$ or approximately n^g possible ways to align..

Dynamic Programming to Find Optimal Sequence Alignment

- In sequence alignment, can piece together optimal prefix alignments to get a ***global*** solution based on optimizing a scoring function (maximizing in this case).
- Can be applied to a wide variety of alignment problems (Max probability through a Markov Chain→Viterbi Algorithm).

The Basic Optimal Alignment Problem has a Complete Algorithmic Solution Using Dynamic Programming

- Define a scoring function
- Find optimal alignment for prefixes of the query and target strings
  - May need to insert gaps to accomplish this
- Extend the process to larger chunks of the problem
  - Dynamic Programming

A Problem

Consider a network of cities connected by some roads.

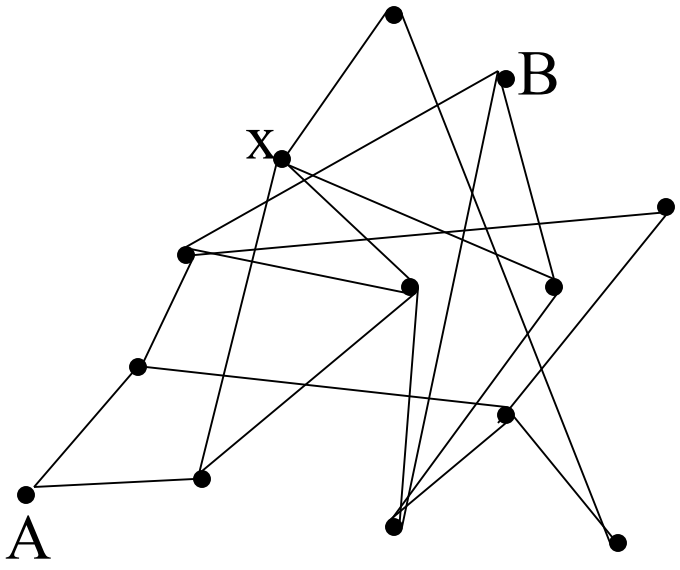
What is the shortest distance from City A to City B ? (optimal solution)

The answer will have the minimum cost function (in this case, distance) of all possible routes

Solution

We know that the distance from A to B d(A,B) is equal to the distance from A to some arbitrary city,x, and the distance from x to B

$$d(A,B)=d(A,x)+d(x,B)$$



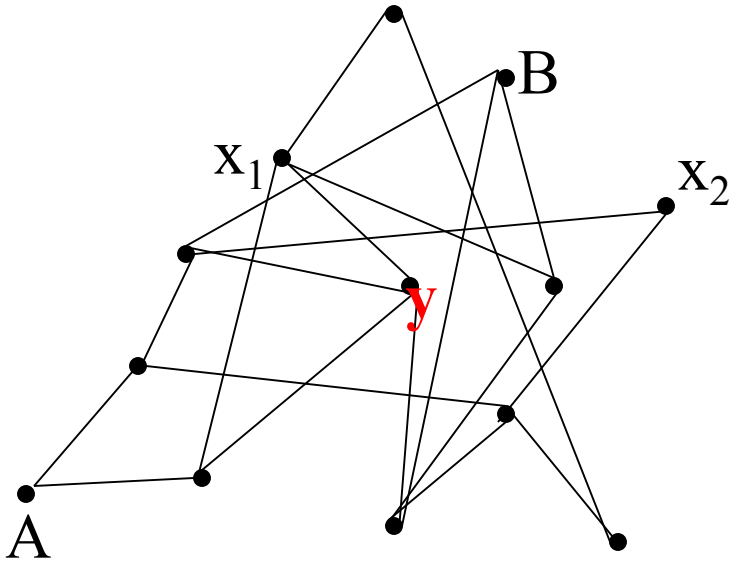
Our problem is that we don't know that the **shortest** path necessarily passes through x

Solution

BUT, we do know that if we look at ***every*** city, the shortest path will pass through ***one*** of them.

The problem, then, has the ***principle of optimality: wherever we start, decisions* ensuing from the first decision are optimal decisions***

Let's pick that optimal one. Call it y and ask the same question



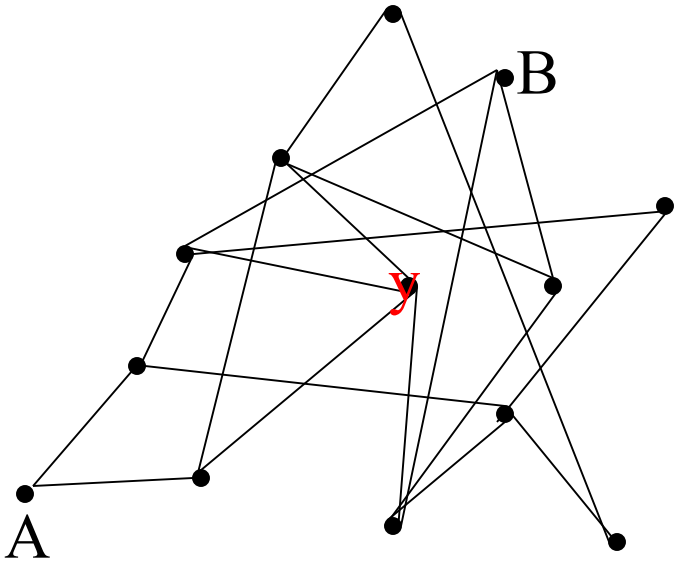
*More abstractly, the decision-making is sometimes called a 'policy'; usually the return from the policy is a scalar

Same Question

What is the shortest distance from A to y, and likewise from y to B.

Same answer: If we look at all the remaining cities, the shortest path will pass through one of them

(likewise for d(y,B))



We have defined a recursive relationship

Same Question

- We can keep asking the same question, and applying the identical answer, till we run out of remaining cities.

- We have glued together optimal solutions to sub-problems by recursively applying the answer to the recurring question.
- This is an exponential exercise

There is a recurrence relationship between a part and all smaller parts. Here is an algorithm:

1. Set starting city as Left endpoint and final city as right endpoint
2. GetDist

Function GetDist

- Exit when down to the smallest ***(always plan your getaway!!)***
- Compute all distances between endpoints passing through all trial cities and find mindist city
- Keep the left endpoint and set the mindist city as a right endpoint
- Set the mindist city as the left endpoint and keep the right endpoint
- GetDist

Recursive Algorithms

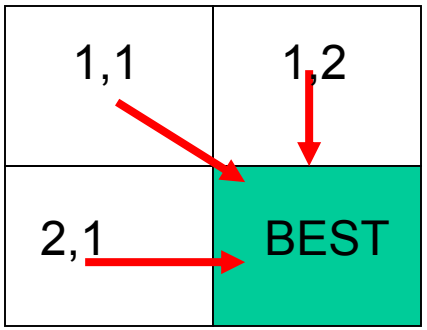
- Again, computations grow exponentially with the number of recursive calls
- But… the number of ***distinct*** recursive calls grows as a polynomial.
- Recursion is thus a "top-down" inefficient solution to the alignment problem since it is exponentially recalculating previously calculated information, which theoretically could be found in polynomial time

Dynamic Programming One Answer….

- Instead of Top-down, build a "Bottom-up" solution for only the elements corresponding to *distinct* recursive calls
- More conceptually complex, ***BUT***……
  - The global solution runs in time $\mathcal{O}(n^k)$ (polynomial time)
  - Works when a problem possesses the principle of optimality.
  - If so, will <u>always</u> give the <u>optimal</u> <u>solution</u> (not a heuristic)

Bottom-Up Computation Matrix Form

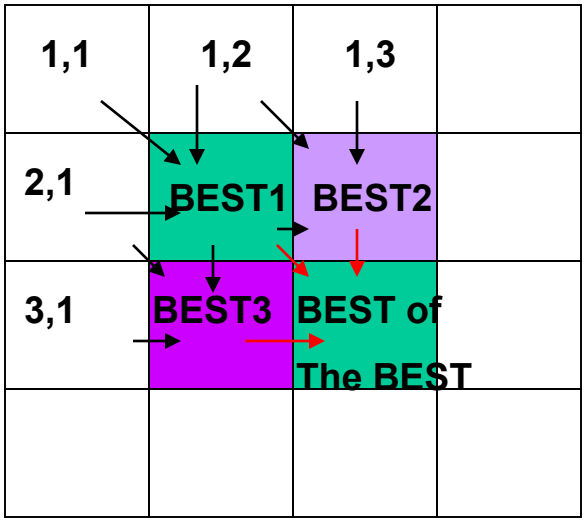
- Start with smallest possible indices (i,j) of the two strings —(1,1) here



- Compute best solution from 3 choices

Bottom-Up Computation Matrix Form

- Increase the size of the problem by incrementing an index and select best of all possible smaller solutions



Best1 = Max[(1,1),(1,2),(2,1)]

Best2 = Max[Best1,(1,2),(1,3)]

Best3 = Max[Best1,(3,1),(2,1)]

Best of the Best =
Max[Best1Best2,Best3]

Dynamic Programming

But because we have already analyzed 1,1 1,2 and 2,1 we don't have to do it again; we just need the best solution to put into 2,2 as an element in the next step.

Dynamic Programming

- Bottom-up computation
- Traceback
  - For each increasing size, keep track of ***which*** of the (3) possible subsolutions was the optimal one

A Popular Scoring Rule for Alignment* on a Matrix

$$A(i,j) = \max \begin{cases} A(i-1,j) & -2 & gap \text{ (deletion)} \\ A(i,j-1) & -2 & gap \text{ (insertion)} \\ A(i-1,j-1) & \pm 1 & \text{+ for match, - for mismatch (replacement)} \end{cases}$$

*We will be working with bases (A,C,G,T) in these examples

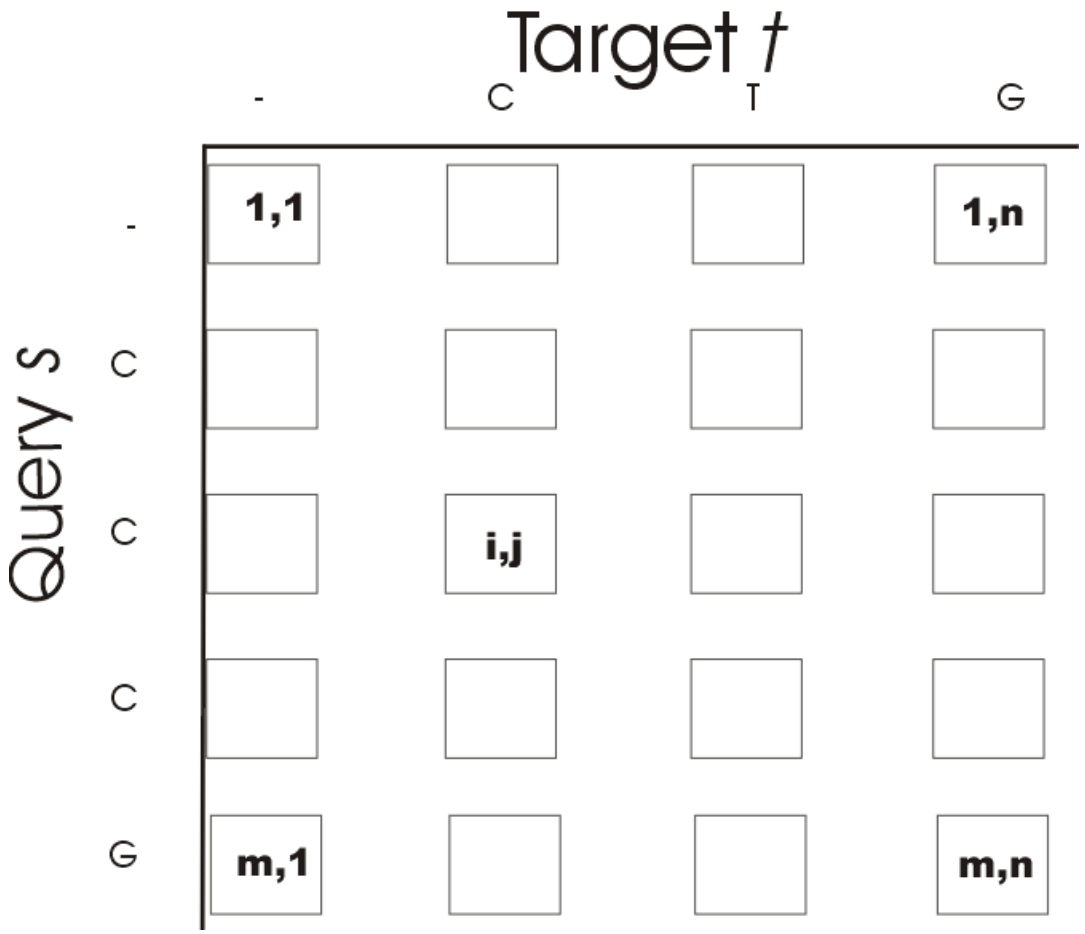
Dynamic Programming

- We are going to look at all possible prefixes in the query (m) against all possible prefixes in the target (n).
- At each step, we will pick out whether we get the best score with an indel, a match, or a replacement, based not on our local score alone, but also in comparison with the cumulative score presented in adjacent cells
- The difficulty of the problem is O(mn)

Global Alignment

Dynamic Programming Algorithm

Needleman-Wunsch

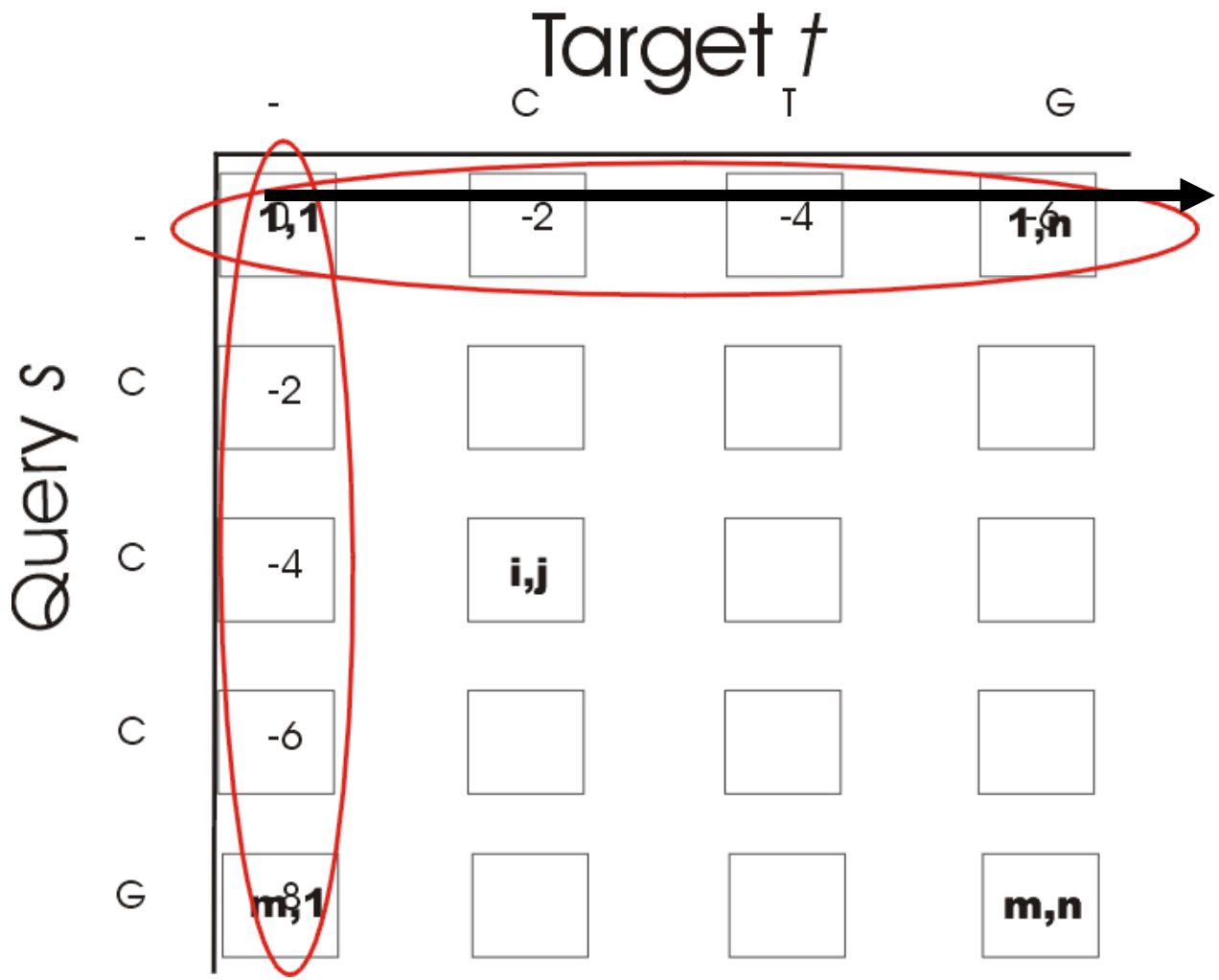


In a global alignment, the idea is to get to the m,n th cell, then trace backward

Global Alignment

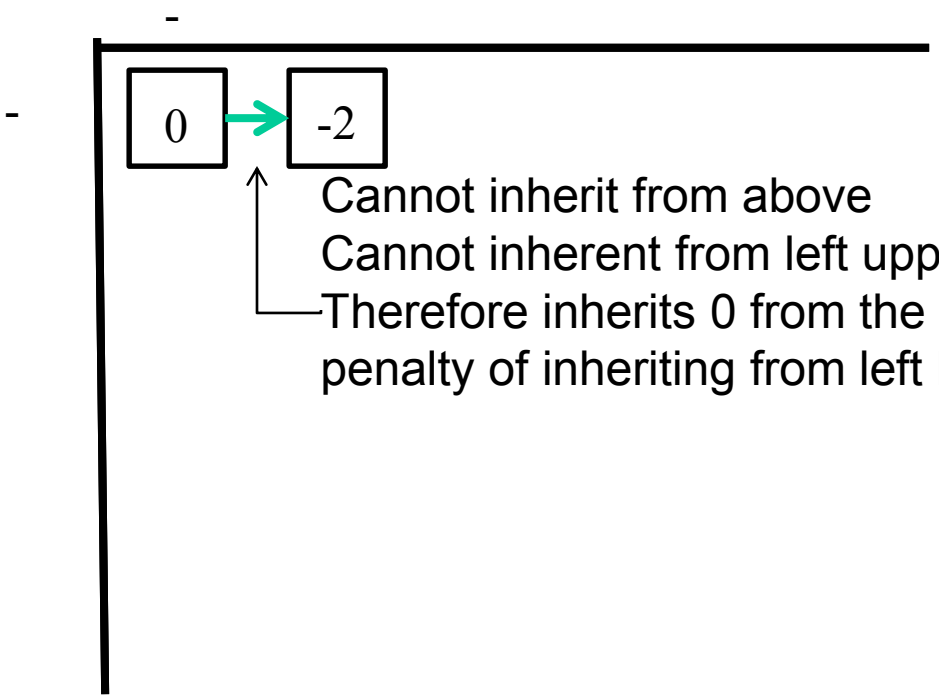
Dynamic Programming Algorithm

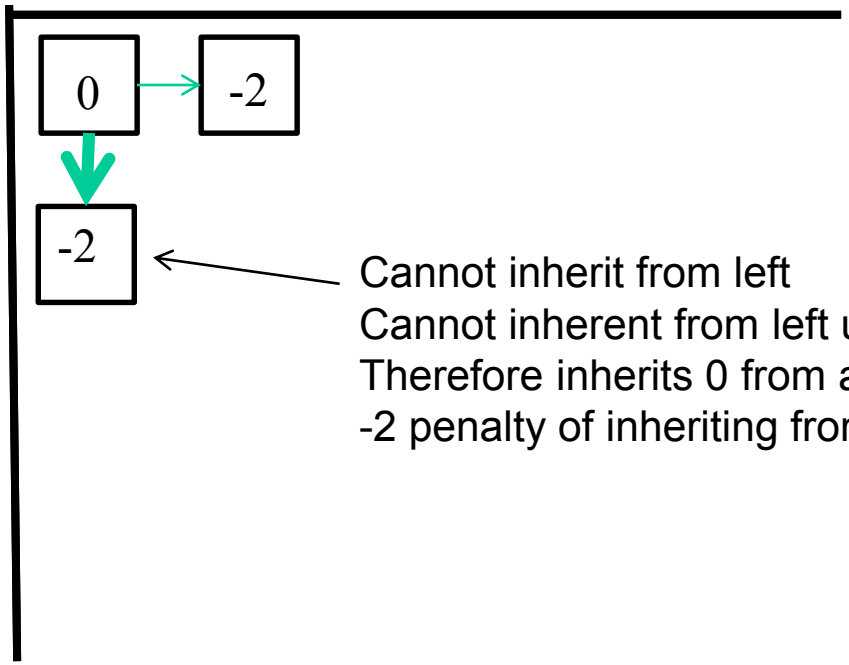
Needleman-Wunsch



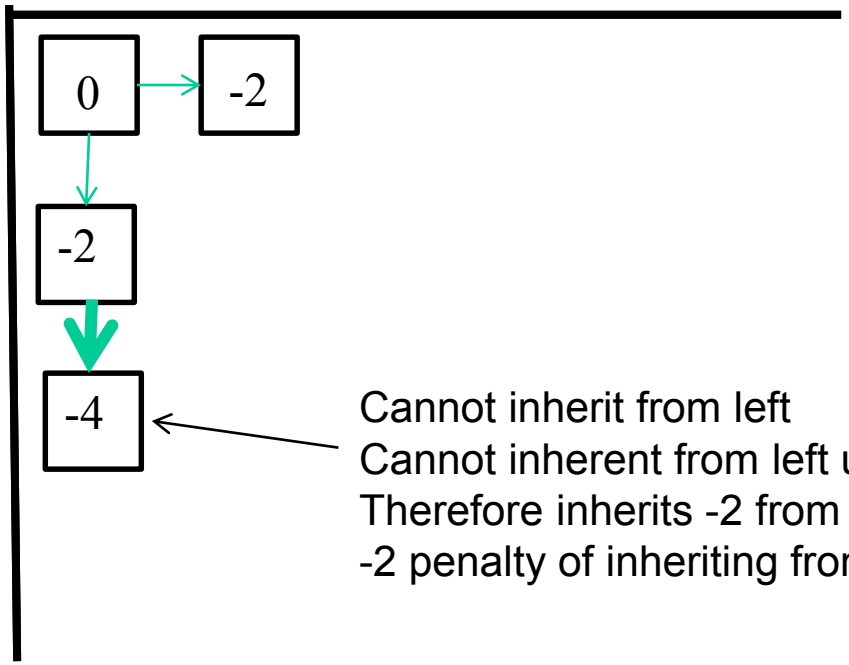
Propagation of horizontal gap penalties.

This is because each target position 'inherits' a gap penalty from the left and each query position 'inherits' a gap penalty from above


-
-
0
-2
Cannot inherit from above
Cannot inherent from left upper diagonal
Therefore inherits 0 from the left and pays the additional -2
penalty of inheriting from left horizontal

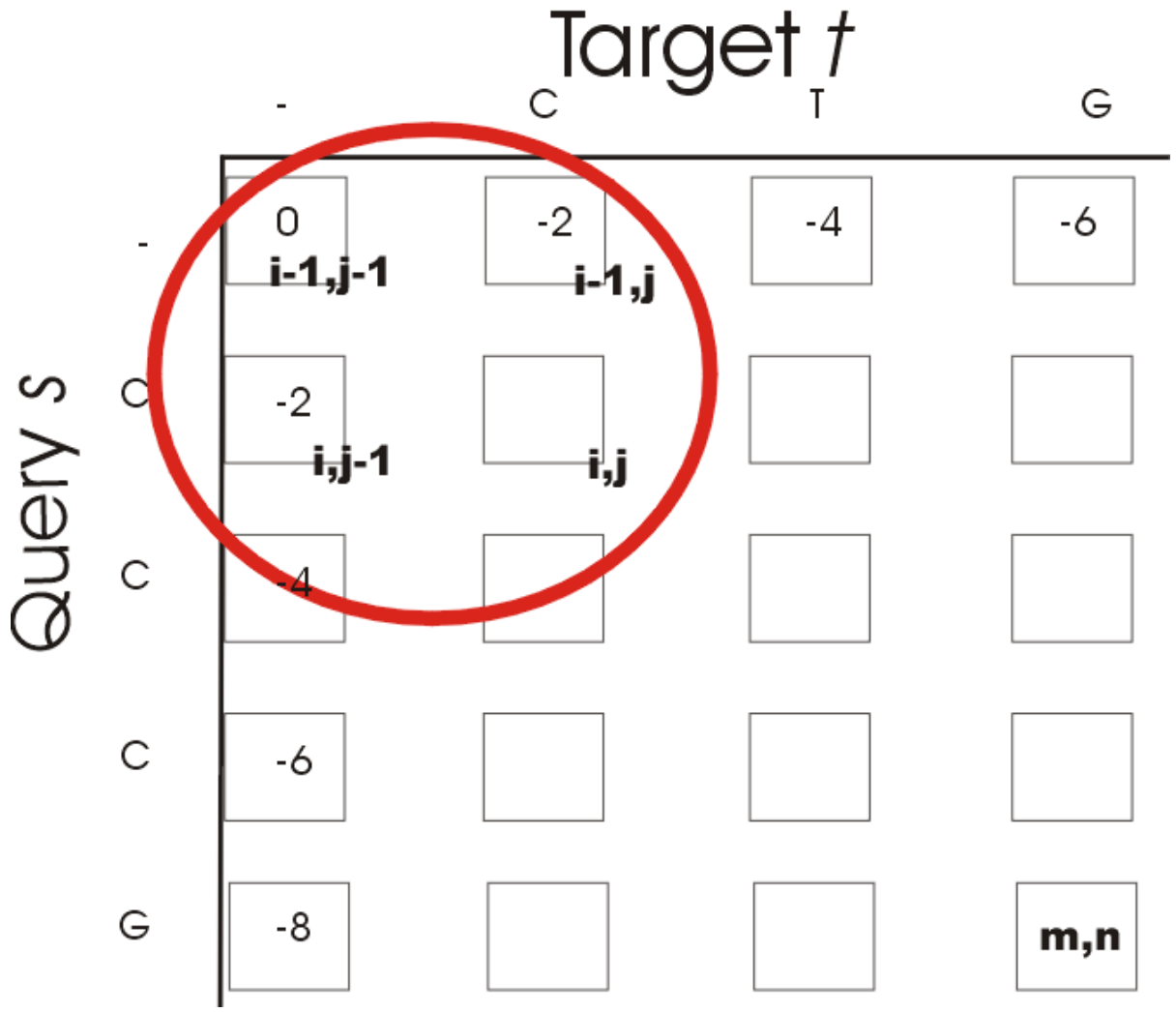


Cannot inherit from left
Cannot inherent from left upper diagonal
Therefore inherits 0 from above and pays the additional -2 penalty of inheriting from above

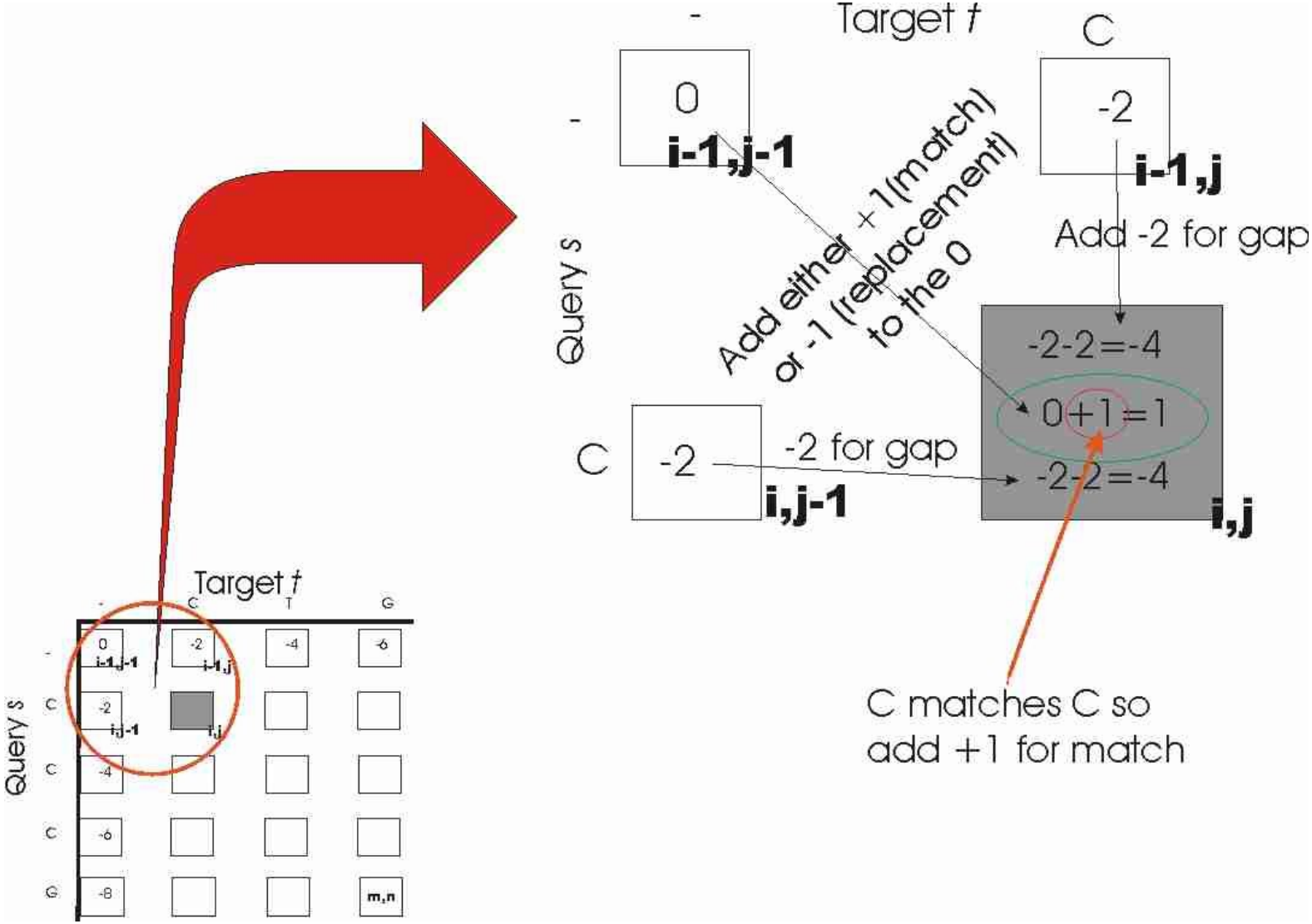


Cannot inherit from left
Cannot inherent from left upper diagonal
Therefore inherits -2 from above and pays the additional -2 penalty of inheriting from above

Global Alignment
Needleman-Wunsch
First prefix is examined

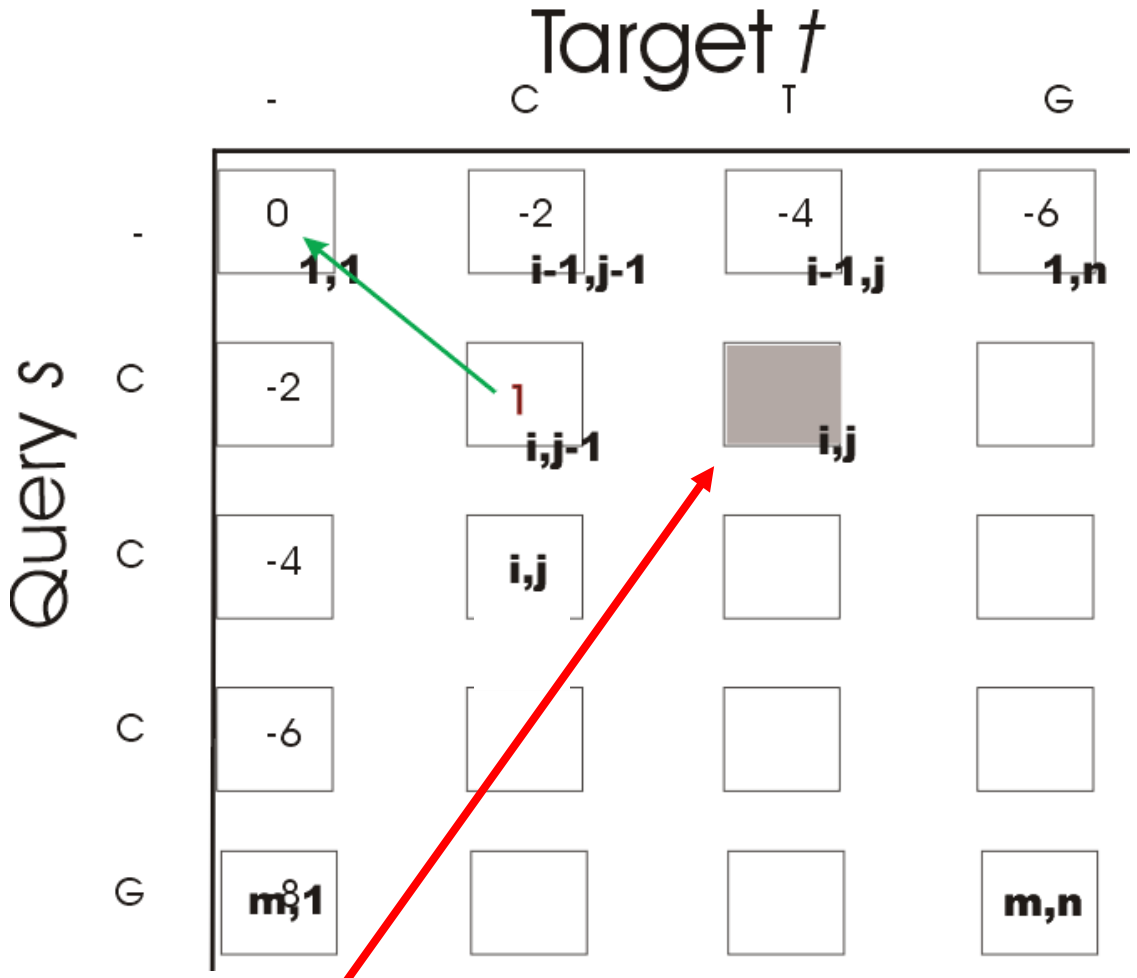


Global Alignment
Needleman-Wunsch
Best score is selected



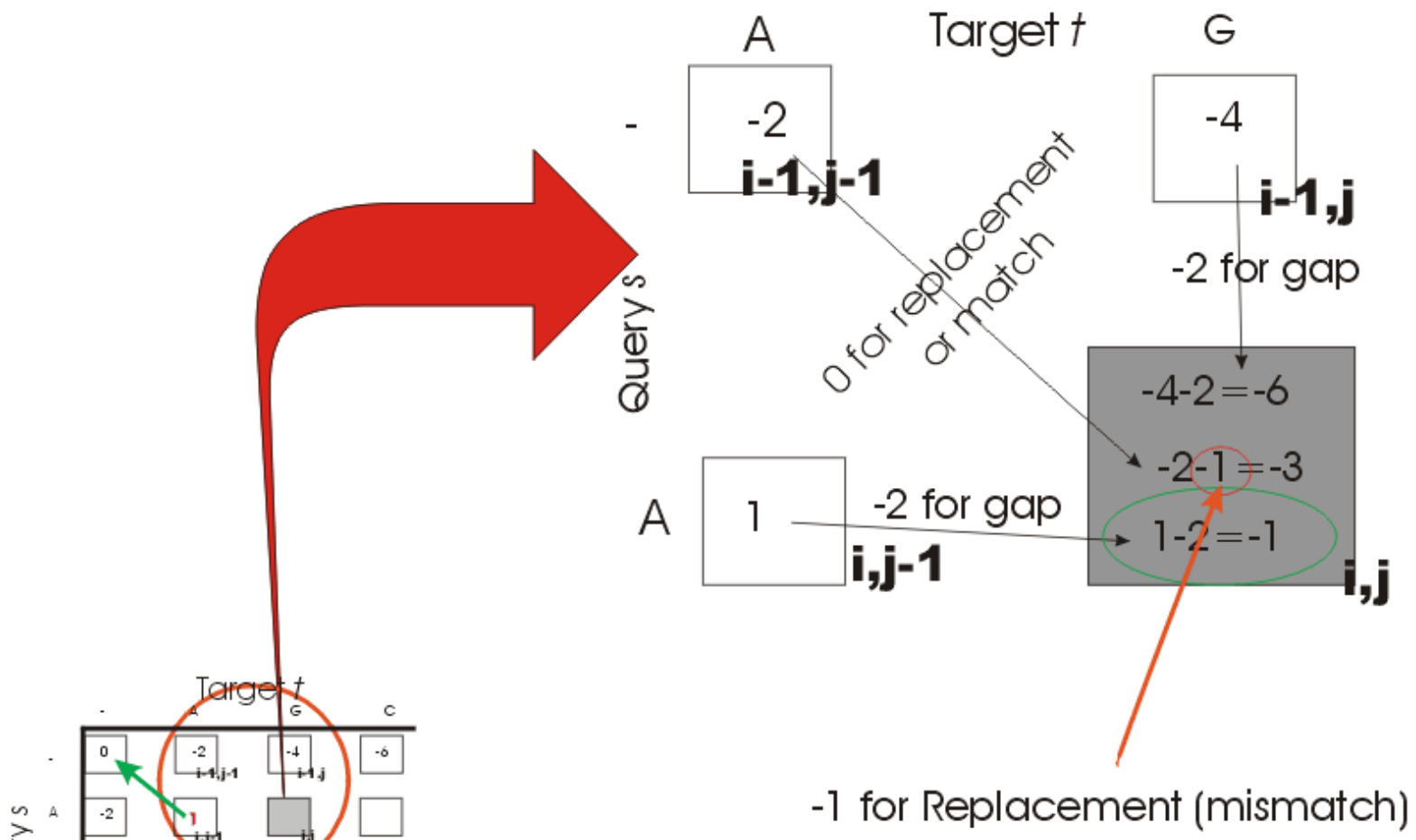
THE TRACEBACK

The path(s) by which the optimum prefix was generated is kept, along with the score itself



Next, a new, adjacent cell is evaluated

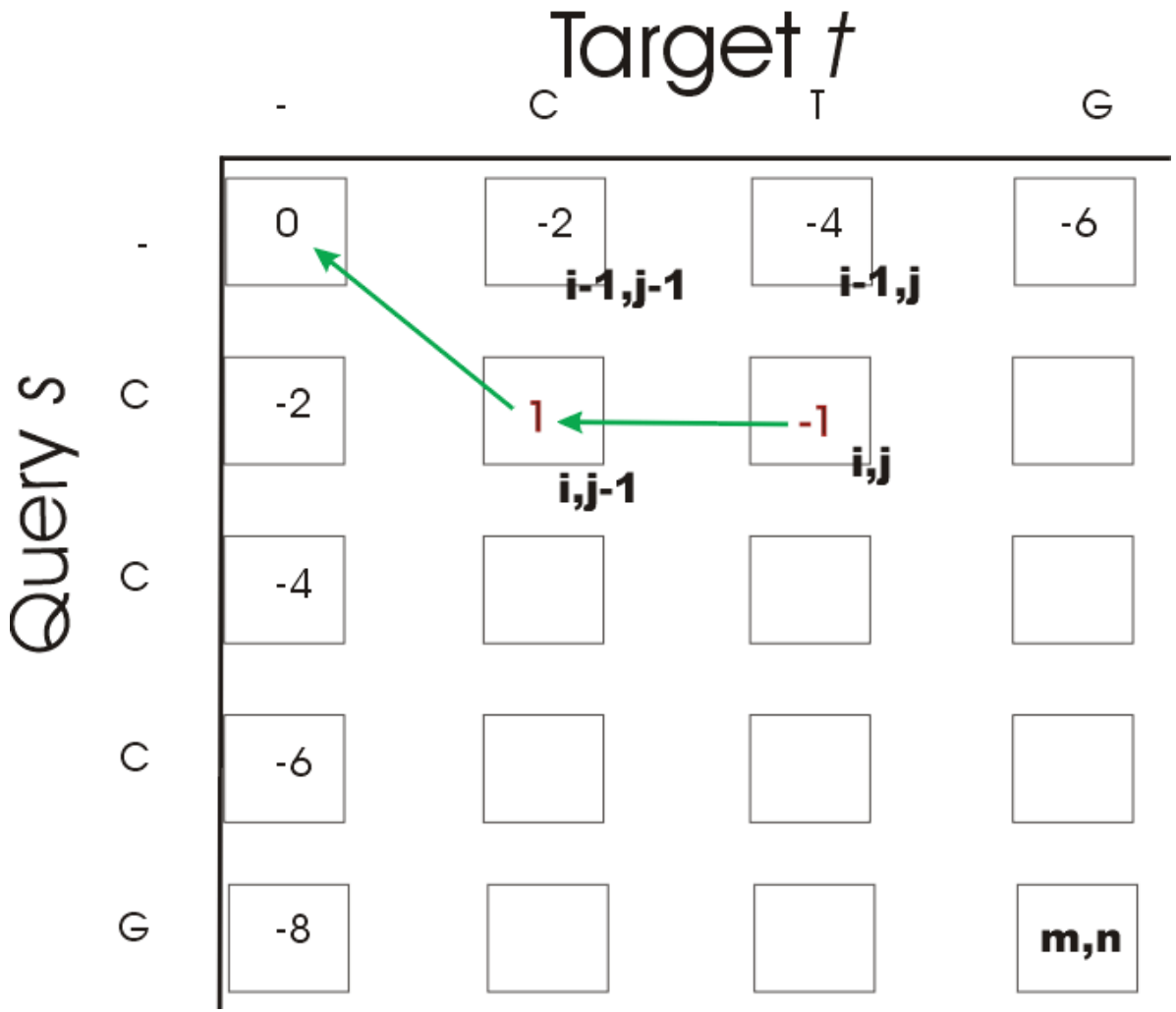
Global Alignment
The next optimum prefix is determined
by finding the best score



In this cell, the best score (-1)
came from a gap in the query

Global Alignment

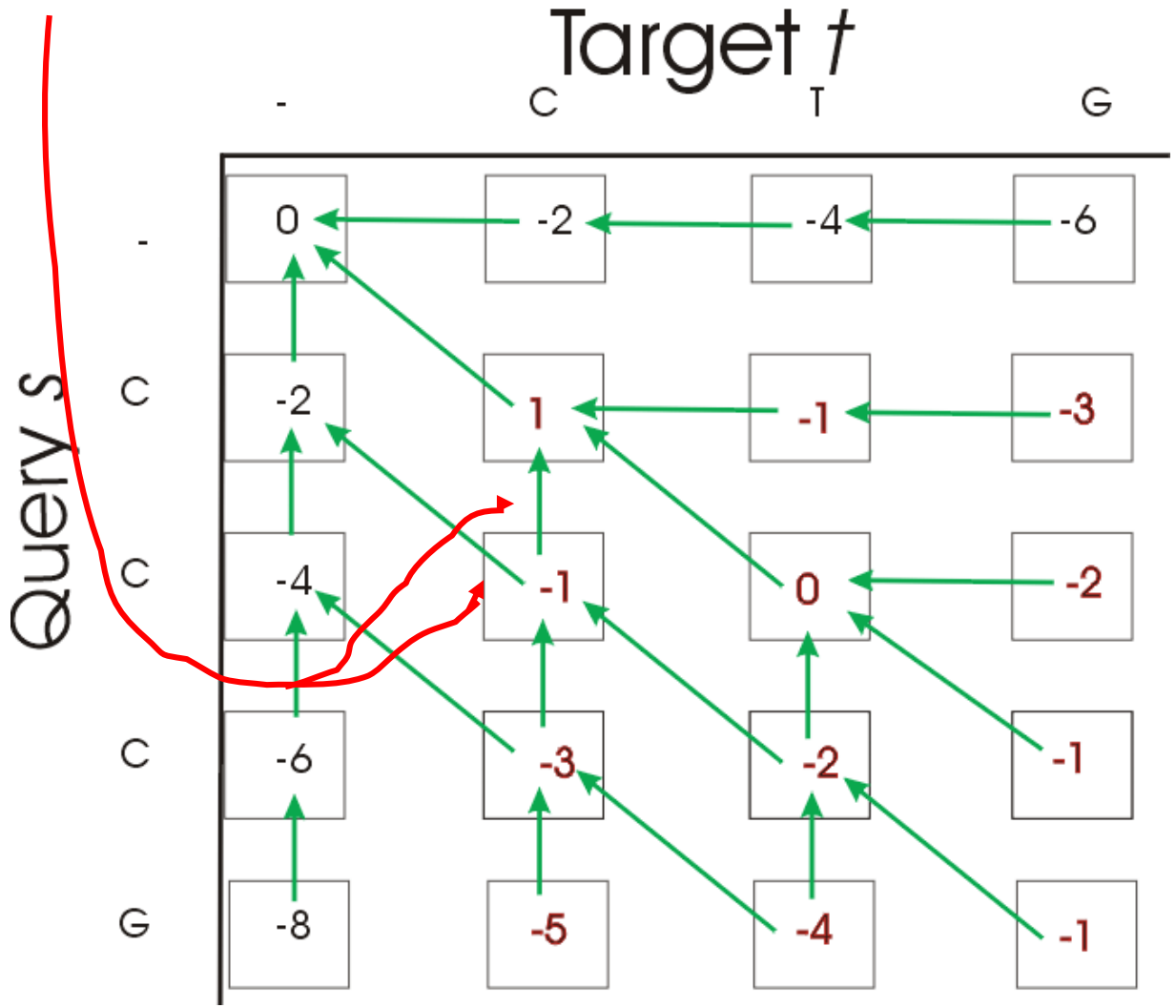
The path and score to that prefix are remembered, as well



Global Alignment

All optimum paths are determined until the m,n *th* position is reached

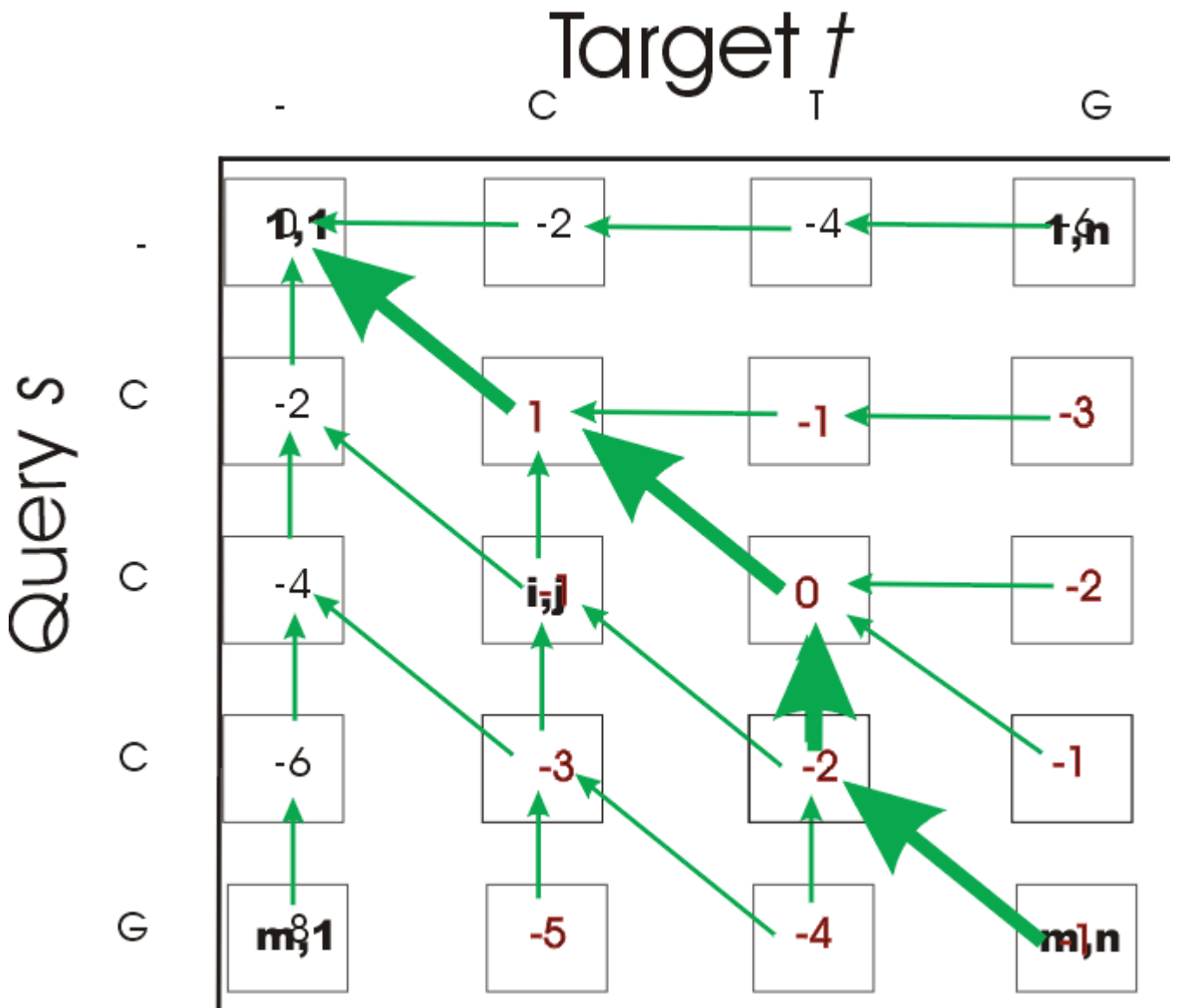
NOTE: If there are two sources of the optimal score in a cell, keep BOTH tracebacks!



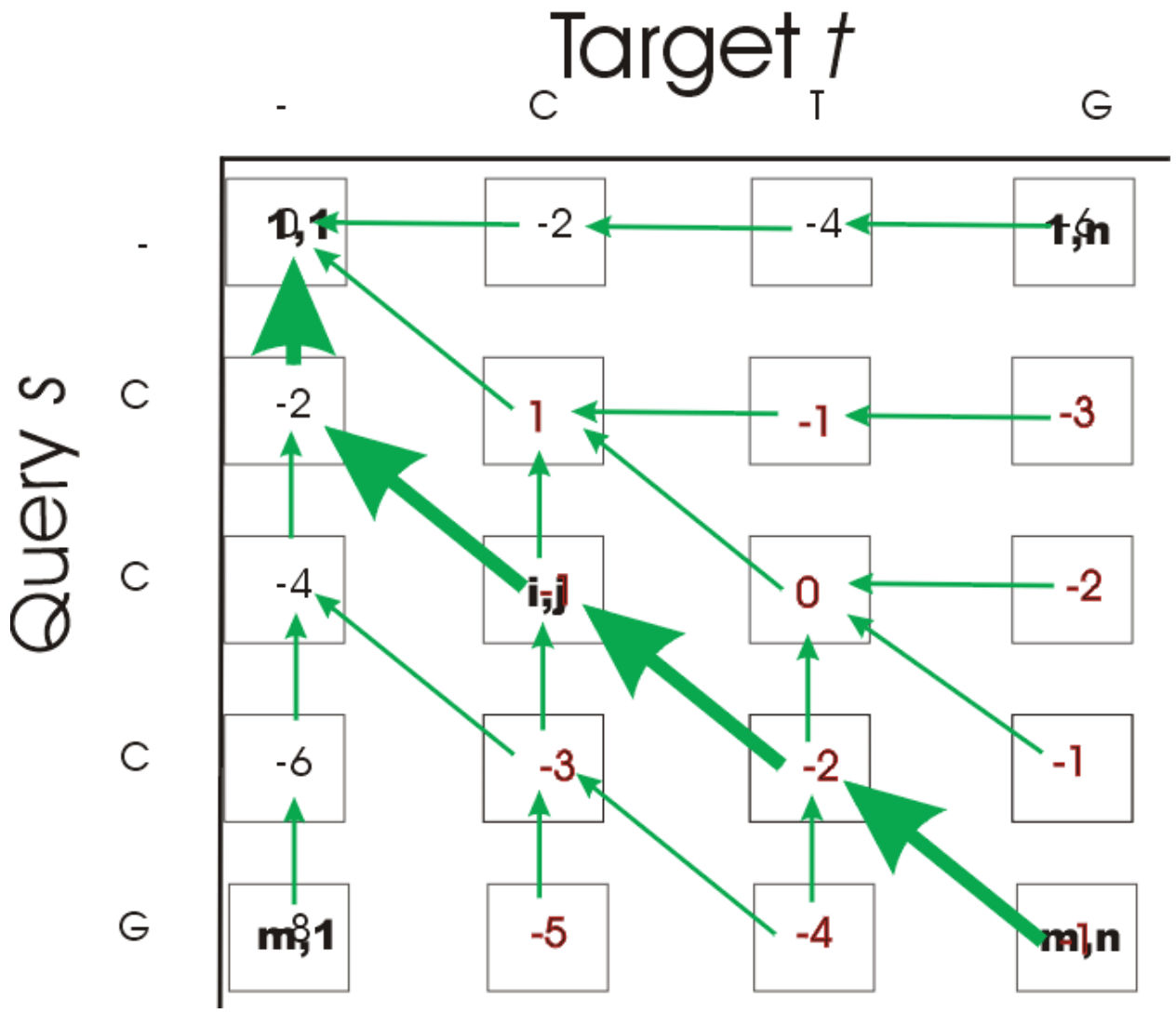
Global Alignment

From the m,n *th* position, the highest scoring continuous path(s) back are determined. This (these) are the optimal alignments.

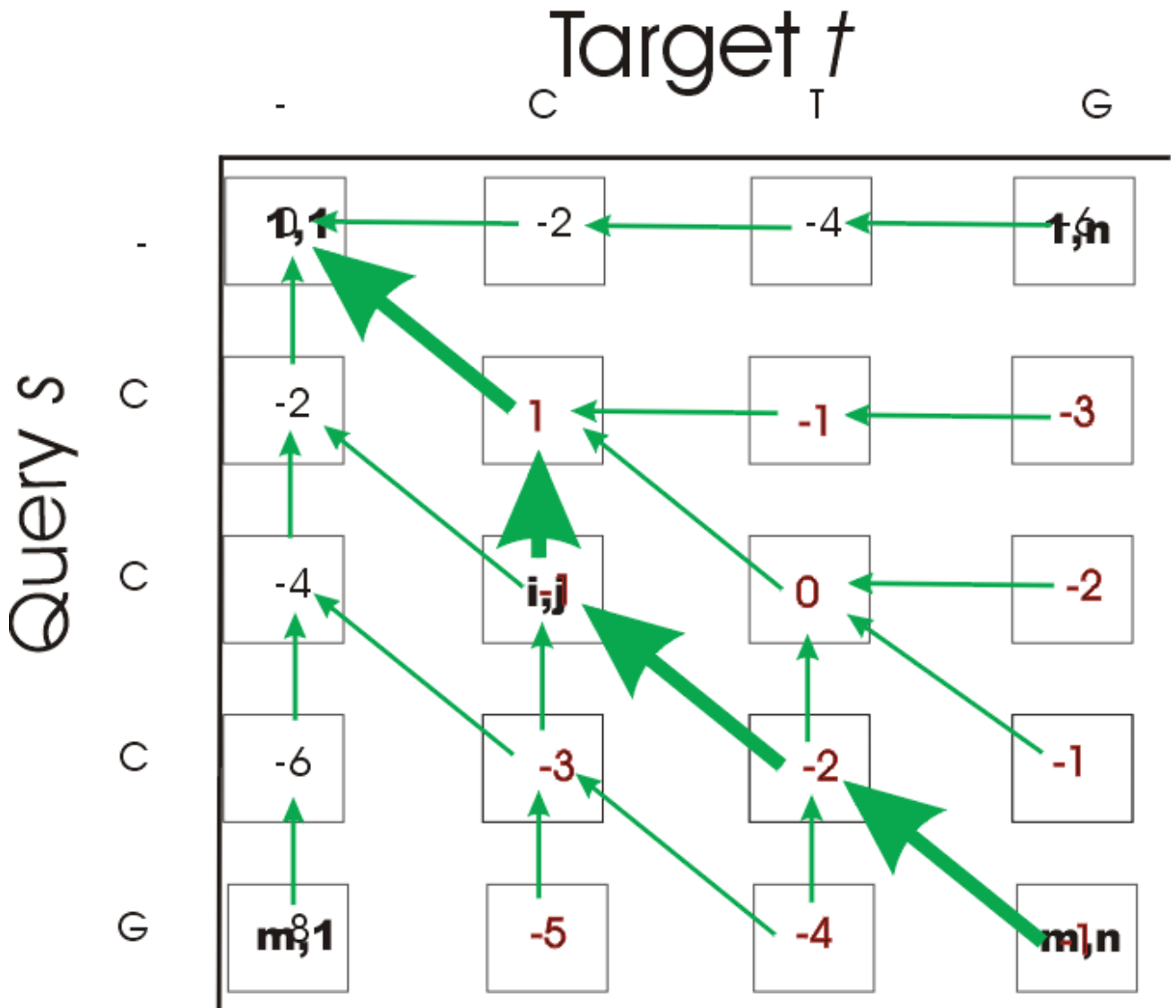
This score is -1



Global Alignment
This score is –1, also



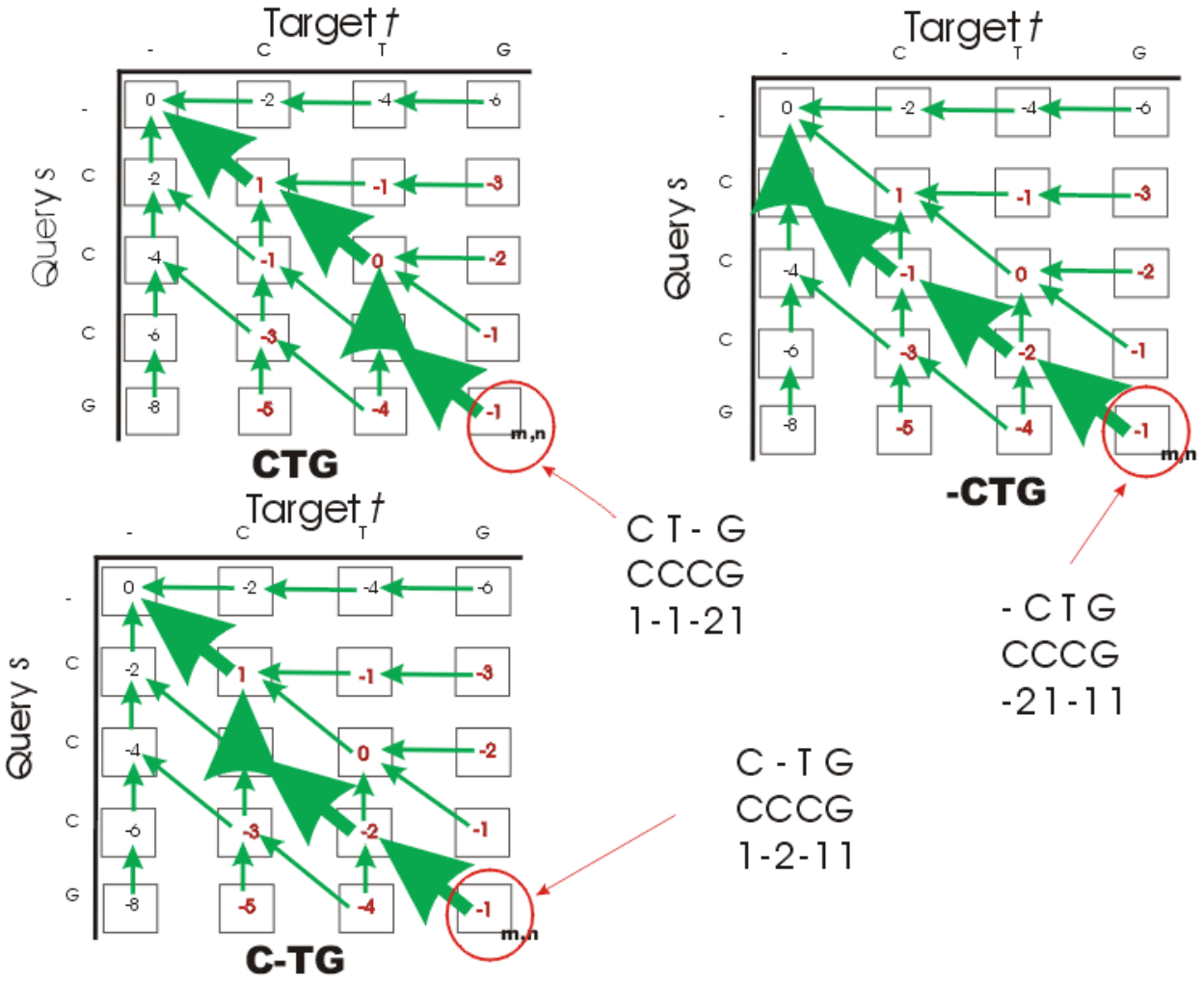
Global Alignment
This score is –1, as well



Global Alignment Constructing the alignment

- Read off the alignment from end to start, beginning with the m,nth cell
  - If leaving a cell diagonally
    - Read off the query and target suffix letters
  - If leaving a cell horizontally
    - Read off the letter in the query
    - A gap in the target
  - If leaving a cell vertically
    - Read off the letter in the target
    - A gap in the query

Global Alignment



Needleman-Wunsch Algorithm

- Runs in polynomial (mn) time
- Runs in mn space as well
  - If made to run in linear space, then finding max score is still easy, BUT finding traceback path(s) is not so easy

LOCAL ALIGNMENT

A ***local alignment*** is an alignment of the query string with a substring[1] of the target string. It is an optimal suffix alignment.

Smith-Waterman Algorithm

For a local alignment

- Finds the highest scoring substrings (suffixes) of the query and target strings
- Waterman: “Fitting one sequence into another”

Local Alignment
Smith-Waterman Algorithm

- Could get a complete solution in $O((mn)^2) \bullet O(mn) = O((mn)^3)$
- S-W runs in O(mn)
- Aligns suffixes instead of prefixes

Local Alignment
Smith-Waterman Algorithm

- There are no initial gaps in the best local alignment, so first row and column have 0's propagated from the origin
- Our general algorithm is modified for a 4th case *i.e.* 0

Local Alignment
Smith-Waterman Algorithm

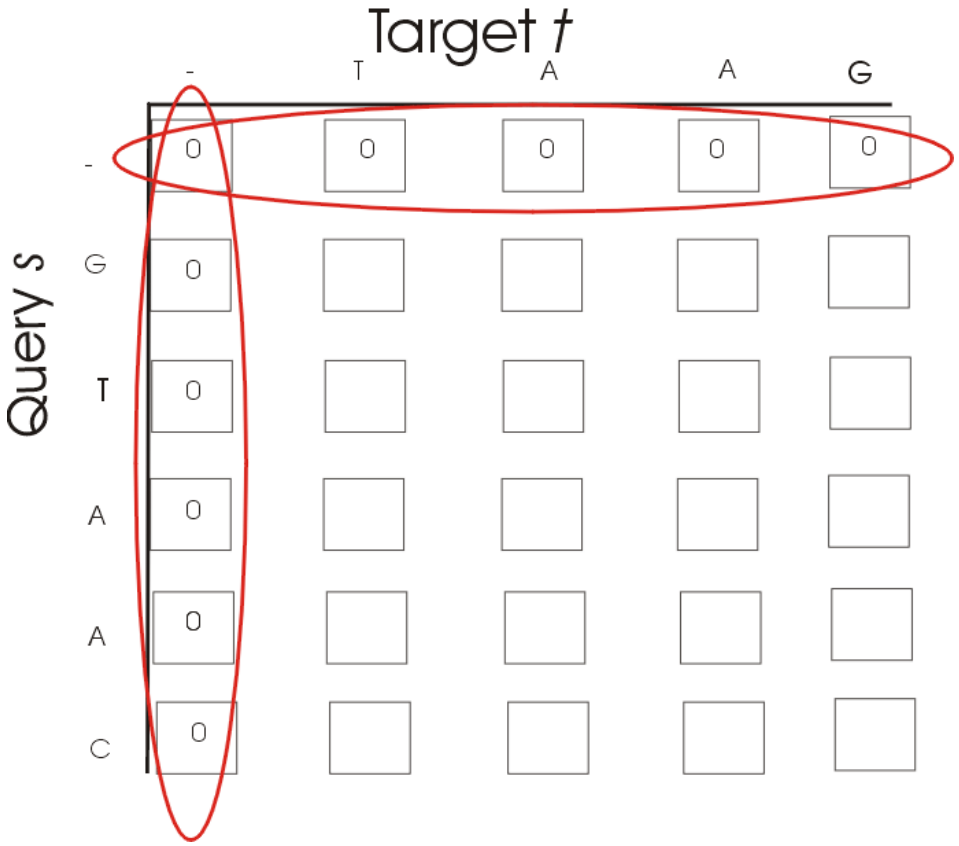
$$A(i,j) = \max \begin{cases} A(i-1,j) & -2 & gap \text{ (deletion)} \\ A(i,j-1) & -2 & gap \text{ (insertion)} \\ A(i-1,j-1) & \pm 1 & + \text{ for match}, - \text{ for mismatch (replacement)} \\ 0 & & \end{cases}$$

Local Alignment
Smith-Waterman Algorithm

- We need to keep track of whether a zero arises from a calculation or from the choice of the fourth case. The zero is needed for as a default max score, but using '0' as a place-marker on the grid is confusing. Might consider a different symbol, say, *, as a place-marker on the trace-back grid when scoring rule defaults to 0 as a max score.
- **NOTE! :** But sometimes the calculation itself results in 0. Must use a 'real' 0, not another symbol, in such a case.

Local Alignment
Smith-Waterman Algorithm

1st row and col are 0's



Why are the first row and column all zeros?

In the Needleman-Wunsch algorithm, they would be increasing negative numbers because of the inheritance penalty.

In the Smith Waterman algorithm, an additional scoring option has been added- a zero (if zero is the maximum score).

So… all negative scores would be replaced by zeros

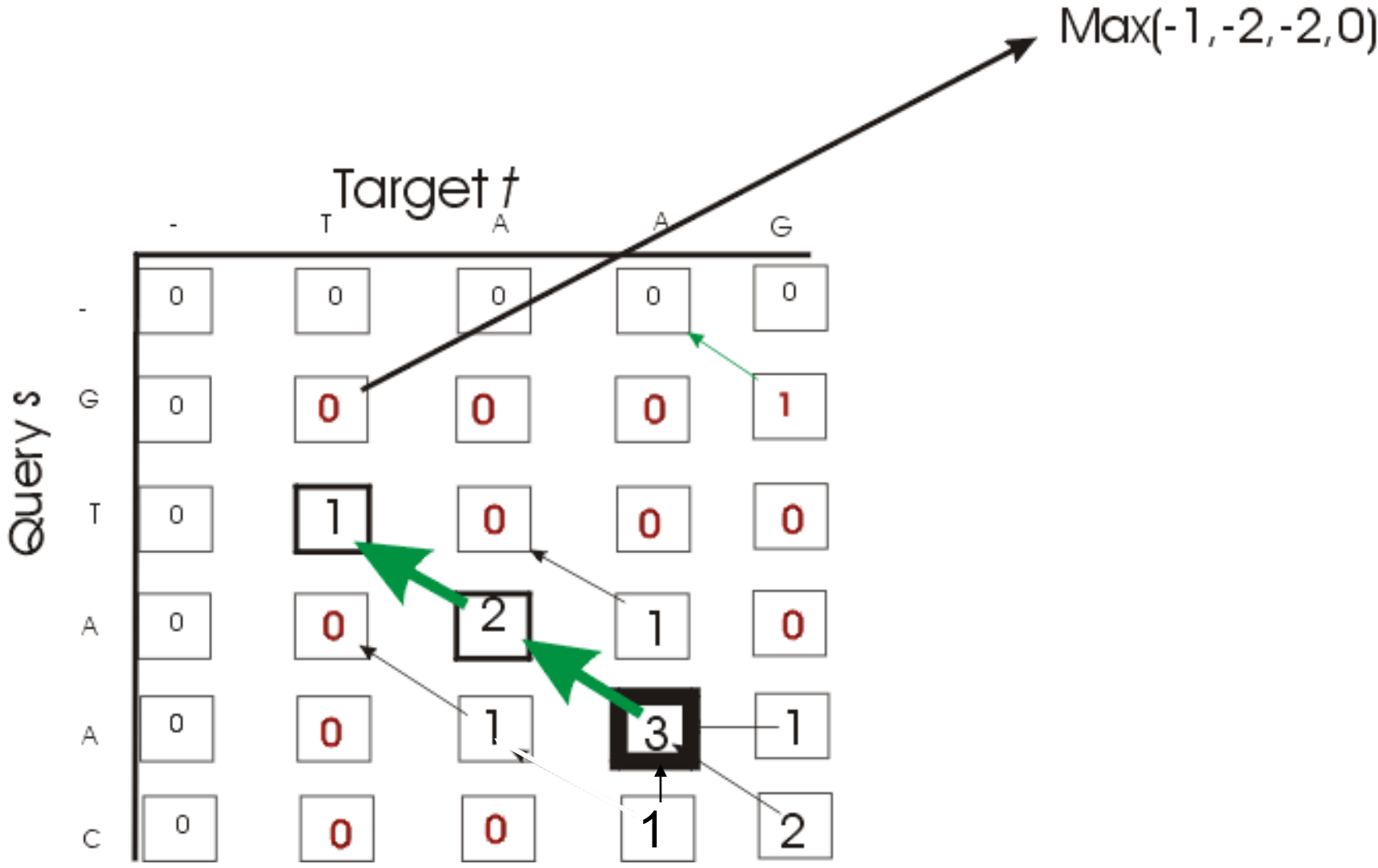
Local Alignment
Smith-Waterman Algorithm
Recipe

- Pad first row and col with 0's
- Apply dynamic programming (modified) algorithm
- Distinguish between selected **0** and computed 0
  - Do traceback arrows with all computed values (incl 0)
  - No traceback arrows from selected **0**
- After matrix is completed, find maximum value
- Trace the path back until there are no arrows out

Local Alignment
Smith-Waterman Algorithm
Nuances

- The max value is the alignment score
- There are sub and superstrings of different scores
- There may be more than one string with the same max value
- There may be other unrelated strings with lower scores which nevertheless might be important to the problem under study

Local Alignment
Smith-Waterman Algorithm



Local Alignment
Smith-Waterman Algorithm
Results

Highest scoring alignment

$$\left.\begin{matrix} T & A & A \\ T & A & A \end{matrix}\right\} = 3$$

Lower scoring superstrings :

$$\left|\begin{matrix} T & A & A & G \\ T & A & A & C \end{matrix}\right\} = 2$$

$$\left|\begin{matrix} T & A & A & G \\ T & A & A & - \end{matrix}\right\} = 1$$

Multiple Sequence Alignment

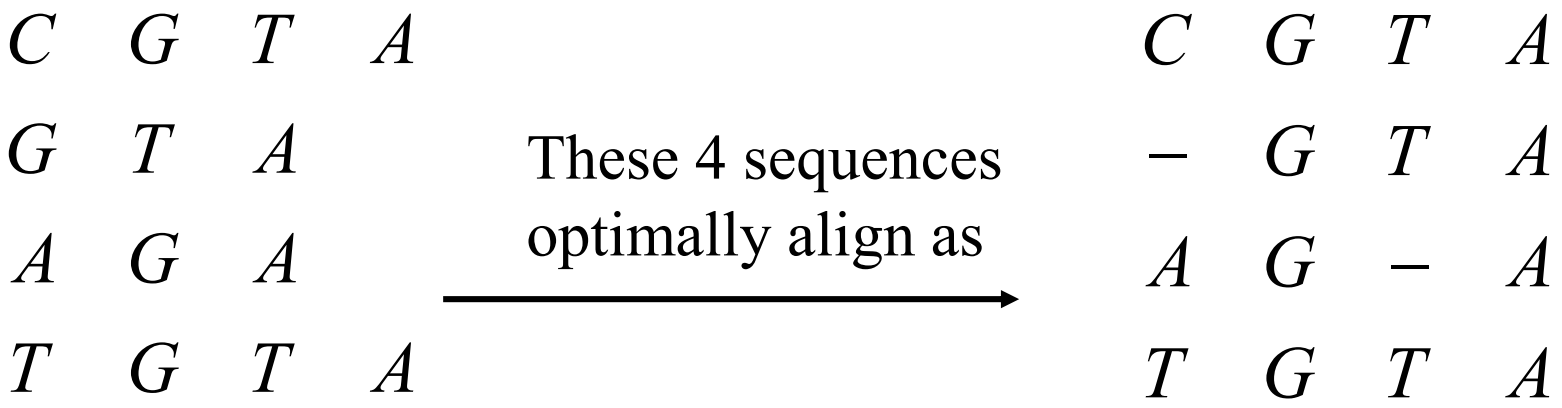
(MSA)

MSA

Based on similarity of the ensemble of sequences, not specific pairs

- Shows patterns
- Discloses families
- Tracks changes (phylogeny)
- Relates mutant genes to wild types or their homologs (Cystic Fibrosis story)

MSA


C G T A
G T A
A G A
T G T A
These 4 sequences
optimally align as
C G T A
– G T A
A G – A
T G T A

MSA Scoring
Sum-of-pairs (SP)

- Go ONLY BY COLUMNS
- Take the sum of the scores of all pairs in each column
  - for 4 rows, there would be

    Score(1^{st},2^{nd})+Score(1^{st},3^{rd})+Score(1^{st},4^{th})
    +Score(2^{nd},3^{rd})+Score(2^{nd},4^{th})+Score(3^{rd},4^{th})
- In your scoring function, gap-gap is given 0

MSA Scoring
Sum of Pairs (SP)

C	G	T	A
G	T	A	–
A	G	A	–
T	G	T	A
–6	0	–2	–7

MSA Score is -15

C	G	T	A
–	G	T	A
A	G	–	A
T	G	T	A
–9	6	–3	6

MSA Score is 0

MSA

This *scoring part* of the algorithm takes

$$n \cdot \frac{k(k-1)}{2}$$

for a problem with k rows where n is the size of the longest string with no gaps. The complexity is $\mathcal{O}(n^2)$ *i.e.*,running in polynomial time

MSA

Now, how will we solve the problem in k dimensions and how hard will it be?

We can certainly apply the dynamic programming algorithm to the k dimensional case.

MSA
Dynamic programming

- The problem now generalizes to a hypermatrix of size n+1 and of dimension k.
- Storage space, then, goes up exponentially with k (without the use of a space-saving heuristic such as the one available for the 2 dimensional case)

MSA

Generalized Dynamic Programming Algorithm

- The space becomes exponential
  - There are n^k hypermatrix entries
- There are 2^{k-1} possibilities for each entry
- Remember the SP scoring scheme is polynomial k^2
- Thus the algorithm for MSA has complexity $\mathcal{O}(k^2 2^k n^k)$

The Bottom-up DP algorithm has therefore become NP-hard

MSA

We need an heuristic!

MSA Heuristic

- 'Star' Alignment (sometimes called merge-alignment) is commonly used.
- Like all heuristics, it is fast but does not guarantee the optimal answer
- It is called 'Star' because one of the k sequences is selected to be 'in the center' to be a basis of comparison to the other k-1 sequences. This might be diagrammed as sequences at the ends of spokes radiating from the center sequence.

Star Alignment

Find the standard alignment scores for each of the (k(k-1)/2) ***pairs of sequences*** and enter them into a k×k matrix.

Consider this set of sequences k=4, of maxlength n=5

s_1 C A T T T

s_2 A T T T A

s_3 A T T G

s_4 G C A T A

Star Alignment

Here are the n(n-1)/2 pairwise alignment scores

$$\left.\begin{matrix} C & A & T & T & T \\ A & T & T & T & A \end{matrix}\right\} s_1 \otimes s_2 = -1$$

$$\left.\begin{matrix} C & A & T & T & T \\ A & T & T & G & - \end{matrix}\right\} s_1 \otimes s_3 = -4 \qquad \left.\begin{matrix} A & T & T & T & A \\ A & T & T & G & - \end{matrix}\right\} s_2 \otimes s_3 = 0$$

$$\left.\begin{matrix} C & A & T & T & T \\ G & C & A & T & A \end{matrix}\right\} s_1 \otimes s_4 = -3 \qquad \left.\begin{matrix} A & T & T & T & A \\ G & C & A & T & A \end{matrix}\right\} s_2 \otimes s_4 = -1 \qquad \left.\begin{matrix} A & T & T & G & - \\ G & C & A & T & A \end{matrix}\right\} s_3 \otimes s_4 = -6$$

Star Alignment

Now make a symmetric matrix of the pairwise alignment combinations, leaving the diagonal blank

	s_1	s_2	s_3	s_4
s_1	·	–1	–4	–3
s_2	–1	·	0	–1
s_3	–4	0	·	–6
s_4	–3	–1	–6	·

Star Alignment

Summing the ROWS of this k×k matrix, s_2 has the best aggregate comparison score

	s_1	s_2	s_3	s_4	
s_1		−1	−4	−3	= −8
s_2	−1		0	−1	= −2
s_3	−4	0		−6	= −10
s_4	−3	−1	−6		= −10

So we pick s_2 to be the center of the star.

Star Alignment

- Find the actual best alignments of each sequence with the center-of-the-star. In this case, s_2 is selected.
- Using the center-of-the-star sequence, s_2 , as the query, align it (global) with each of the remaining sequences, yielding n-1 best alignments.

Star Alignment

Get the optimal pairwise alignments using s_2 as the query

$$\begin{array}{lccccc} s_2 & A & T & T & T & A \\ s_1 & C & A & T & T & T \end{array} \rightarrow \begin{array}{cccccc} - & A & T & T & T & A \\ C & A & T & T & T & - \end{array}$$

$$\begin{array}{lccccc} s_2 & A & T & T & T & A \\ s_3 & A & T & T & G & - \end{array} \rightarrow \begin{array}{ccccc} A & T & T & T & A \\ A & T & T & G & - \end{array}$$

$$\begin{array}{lccccc} s_2 & A & T & T & T & A \\ s_4 & G & C & A & T & A \end{array} \rightarrow \begin{array}{ccccc} A & T & T & T & A \\ G & C & A & T & A \end{array}$$

Final Product of the Star Alignment

Beginning with s_2, merge the optimal alignments, but
KEEP ALL GAPS

s_2	$-$	A	T	T	T	A
s_1	C	A	T	T	T	$-$
s_3	$-$	A	T	T	G	$-$
s_4	$-$	G	C	A	T	A

Star Alignment Complexity

- To build the table and get center of star:
  $\mathcal{O}(k^2n^2)$
- To do the MSA: $\mathcal{O}(kn^2+k^2l)$ (l is length after gaps)
  - $\mathcal{O}(kn^2)$for pairwise alignments
  - $\mathcal{O}(k^2l)$ for assembling sequences

Can be optimized to $\mathcal{O}(kn+kl)$