# 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

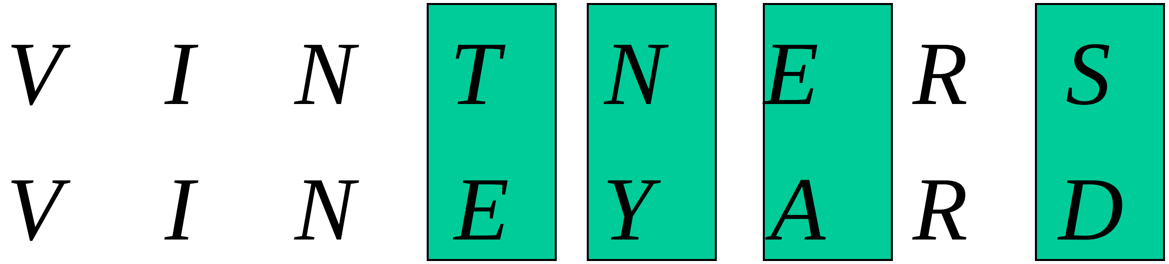
A metric $\rho : X \rightarrow \Re$ is a function with the following properties for $a,b,c \in \Re$

- $P(a) \in \Re$, $\rho \geq 0$ *(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* |
| *V* | *I* | *N* | *E* | *Y* | *A* | *R* | *D* |

Edit Distance = 4

# 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<br>(replacement) | -1 |
| Align with gap<br>(insertion or deletion)<br>Called “Indels” by Waterman | -2 |

# Similarity Scoring of an Alignment

## Example of Two of 6 Possible Alignments

$$\begin{array}{cccccc}
A & T & G & C & A & T \\
C & T & - & G & C & T = -3 \\
\hline
-1 & 1 & -2 & -1 & -1 & 1
\end{array}$$

$$\begin{array}{cccccc}
A & T & G & C & A & T \\
C & T & G & C & - & T = 1 \\
\hline
-1 & 1 & 1 & 1 & -2 & 1
\end{array}$$

# 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

$$\begin{array}{cccccccccc} A & T & G & A & T & A & C & C & C & T \\ T & T & G & - & T & A & C & G & - & T \end{array}$$

LOCAL

$$\begin{array}{cccccccccc} A & T & G & A & T & A & C & C & C & T \\ & T & G & A & A & A & G & G & & \end{array}$$

# Optimal Global Alignments

$$\begin{array}{ccccccl} A & T & G & C & A & T & \\ C & T & - & G & C & T & = -3 \\ \cline{1-6} -1 & 1 & -2 & -1 & -1 & 1 & \end{array}$$

$$\begin{array}{ccccccl} A & T & G & C & A & T & \\ C & T & G & C & - & T & = 1 \\ \cline{1-6} -1 & 1 & 1 & 1 & -2 & 1 & \end{array}$$

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

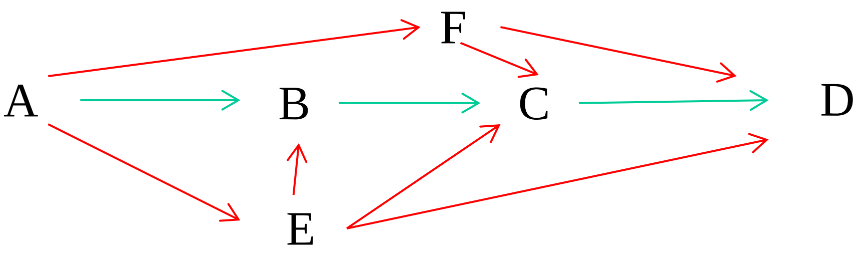
- In sequence alignment, can piece together optimal prefix alignments to get a ***global*** solution based on optimizing a scoring function (maximizing in this case).
- Can be applied to a wide variety of alignment problems (Max probability through a Markov Chain→Viterbi Algorithm) provided the have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

# 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

# Principle of Optimality

The problem must have the *principle of optimality* for a valid dynamic Consider a network of cities connected by some roads.



OPTIMALITY: If the optimal (*sci* shortest) path from A to D passes through B and C
then the optimal path from B to D must also pass through C

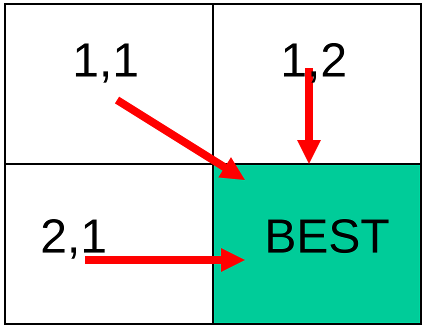
# Dynamic Programming

Conceptually complex, ***BUT*………**

- The solution runs in time $\mathcal{O}(n^k)$ (polynomial time)
- Works when a problem possesses the principle of optimality.
- If so, will always give the optimal solution (not a heuristic)

# Computation -Matrix Form

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



1,1

1,2

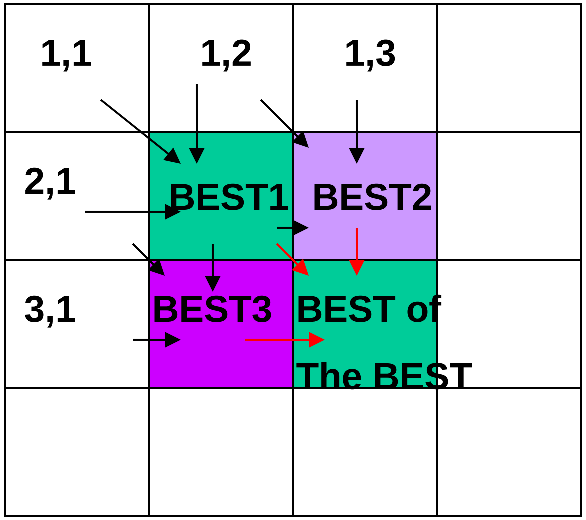
2,1

BEST

- 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


1,1
1,2
1,3
2,1
BEST1
BEST2
3,1
BEST3
BEST of
The BEST

**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. (because of optimality)

# 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

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

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

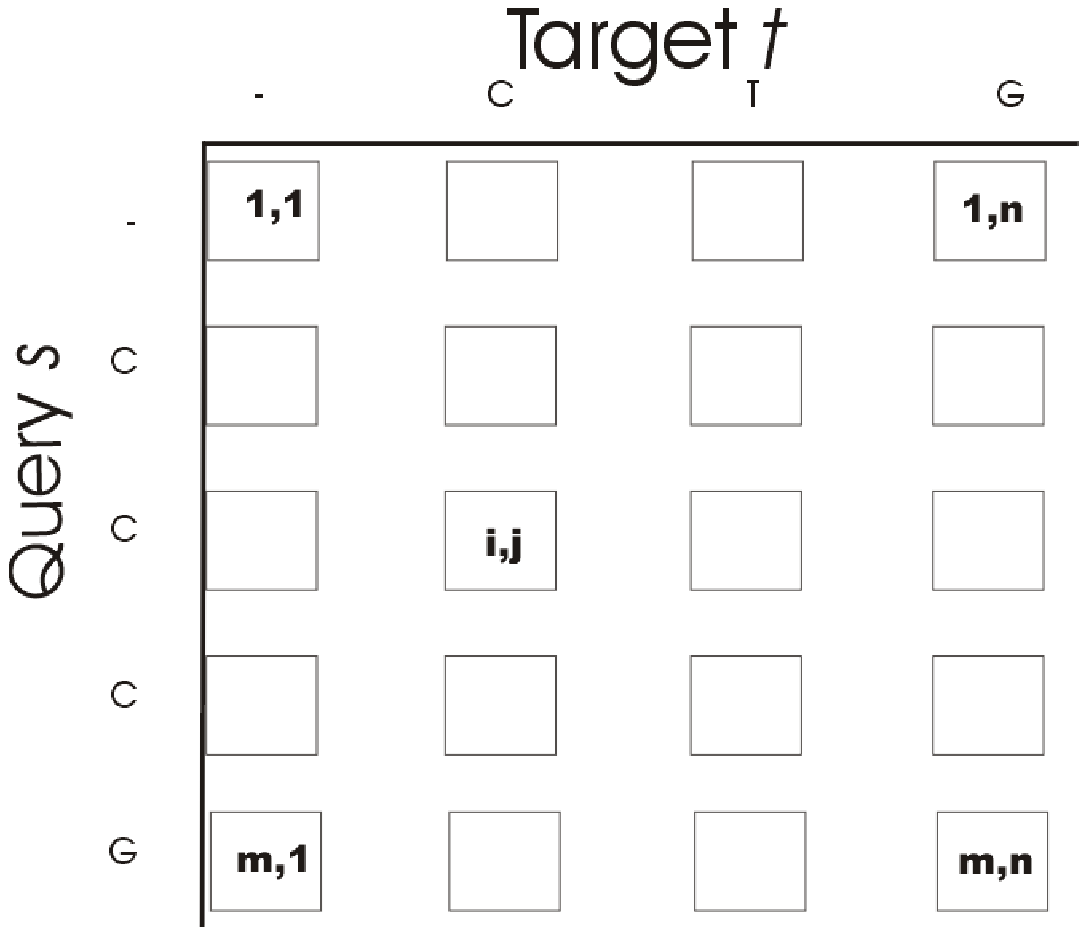
# Dynamic Programming

- 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



Target *t*

Query *s*

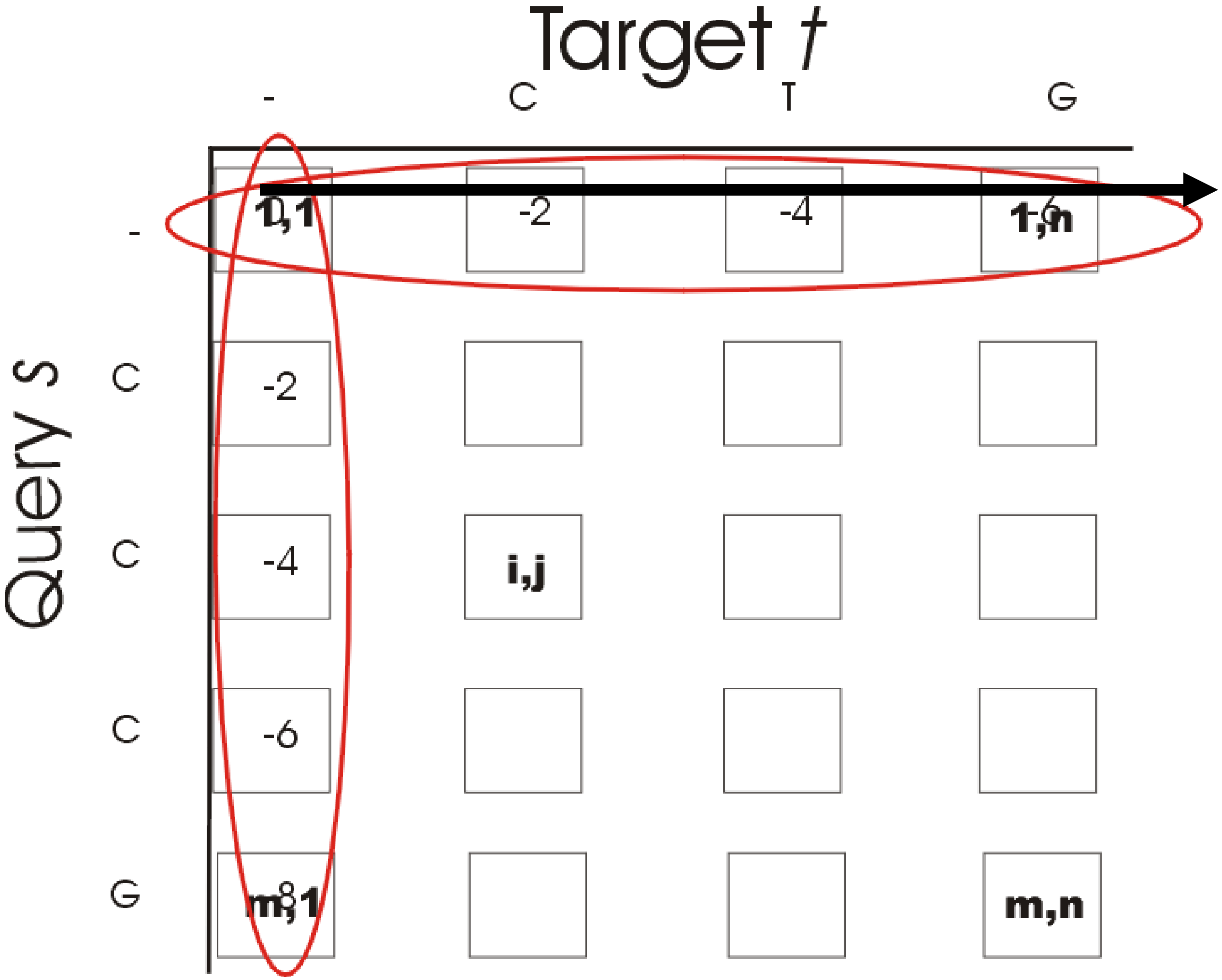
| | - | C | T | G |
|---|---|---|---|---|
| - | **1,1** | | | **1,n** |
| C | | | | |
| C | | **i,j** | | |
| C | | | | |
| G | **m,1** | | | **m,n** |

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

# Global Alignment

## Dynamic Programming Algorithm

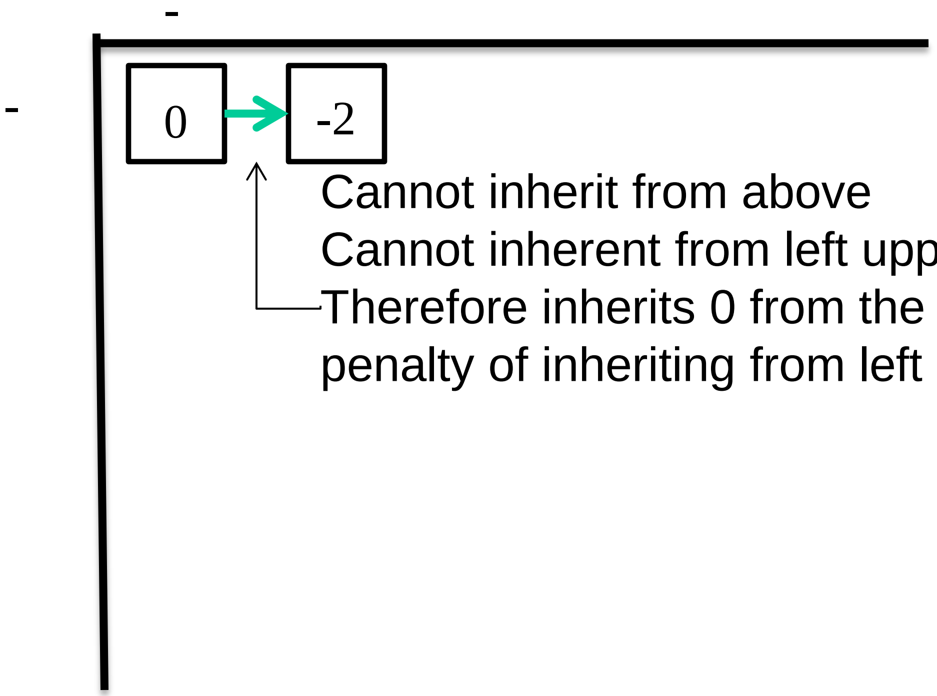
### Needleman-Wunsch



Target *t*
-
C
T
G
Query *s*
-
C
C
C
G
0
1,1
-2
-4
-6
1,n
-2
-4
i,j
-6
-8
m,1
m,n

Propagation of horizontal gap penalties.

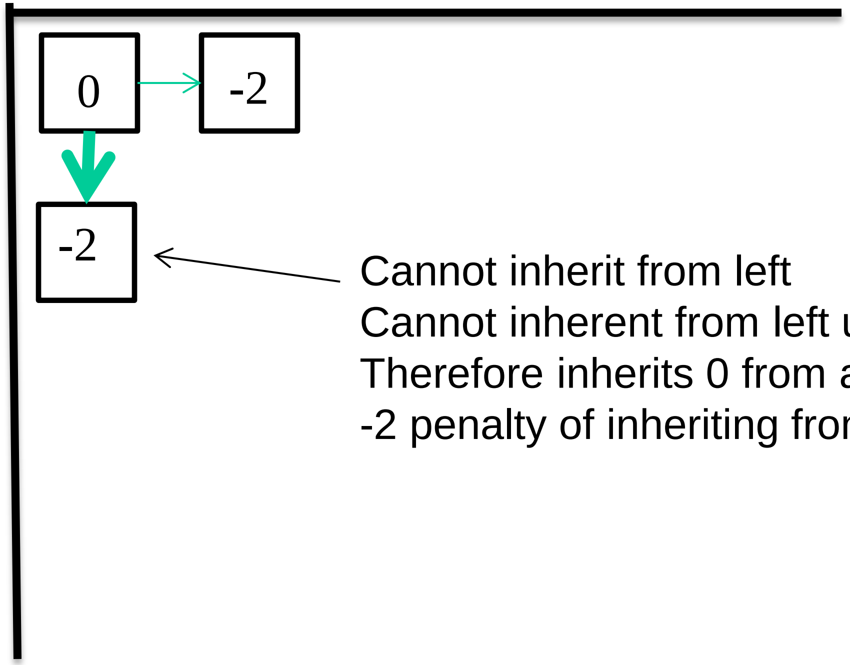
This is because each target position ‘inherits’ a gap penalty from the left and each query position ‘inherits’ a gap penalty from above



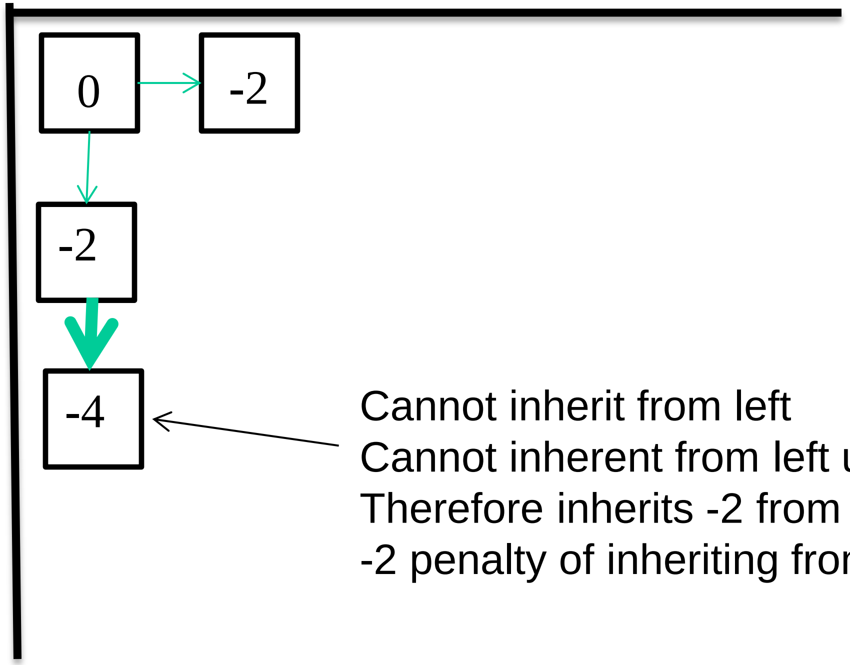
Cannot inherit from above

Cannot inherent from left upper diagonal

Therefore inherits 0 from the left and pays the additional -2 penalty of inheriting from left horizontal

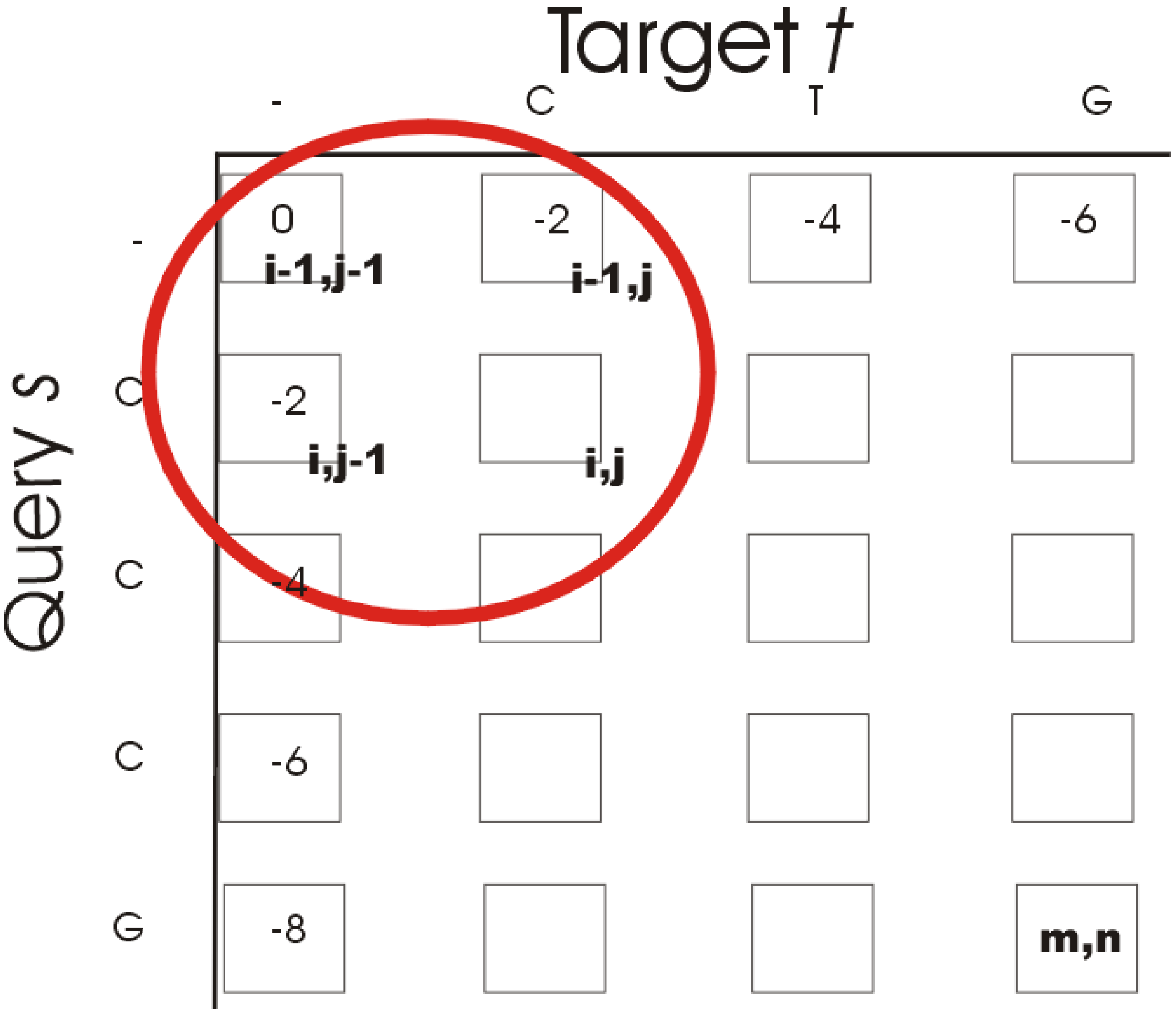


Cannot inherit from left
Cannot inherent from left upper diagonal
Therefore inherits 0 from above and pays the additional
-2 penalty of inheriting from above



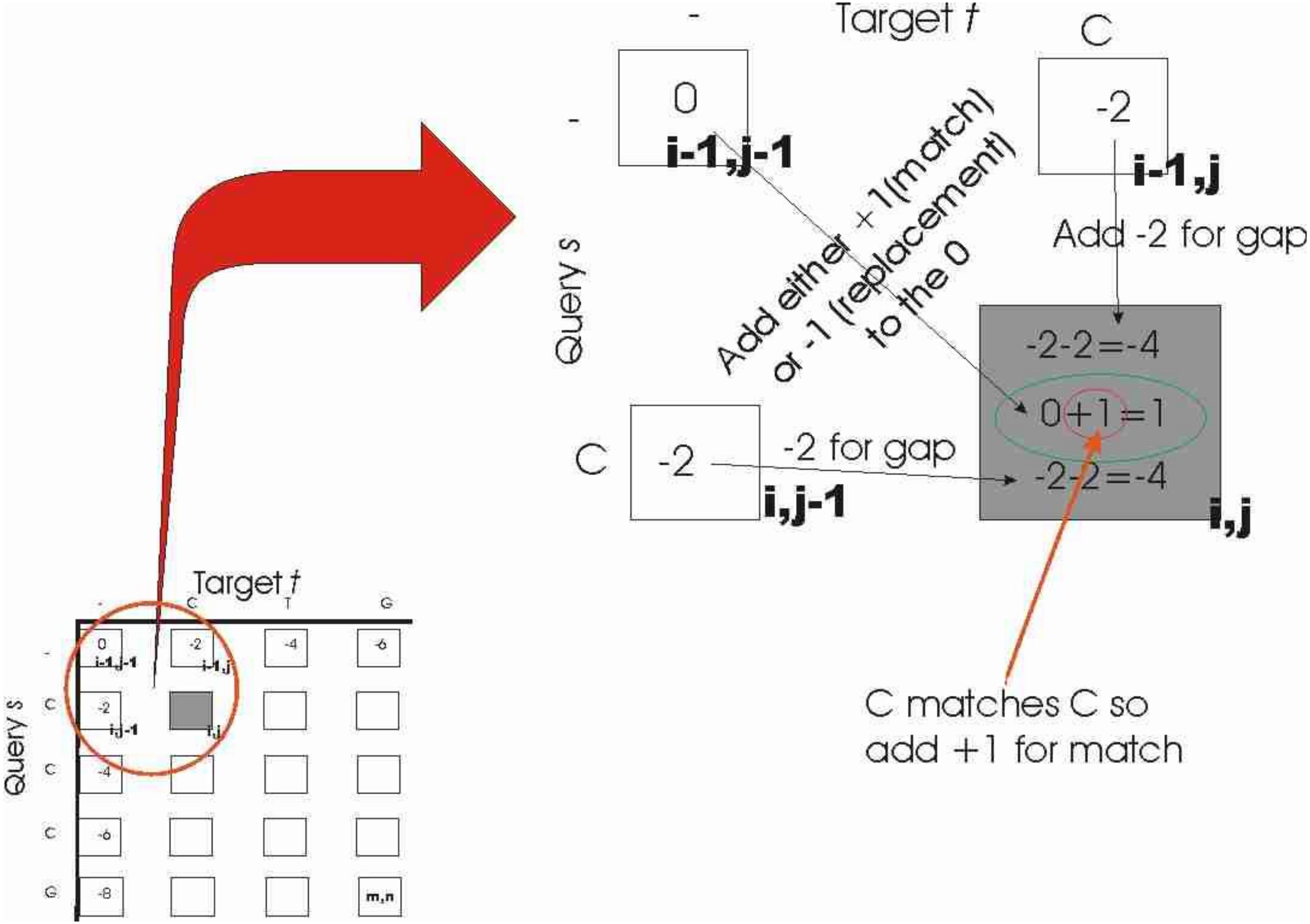
Cannot inherit from left
Cannot inherent from left upper diagonal
Therefore inherits -2 from above and pays the additional
-2 penalty of inheriting from above

# Global Alignment
# Needleman-Wunsch
# First prefix is examined


Target *t*
Query *s*

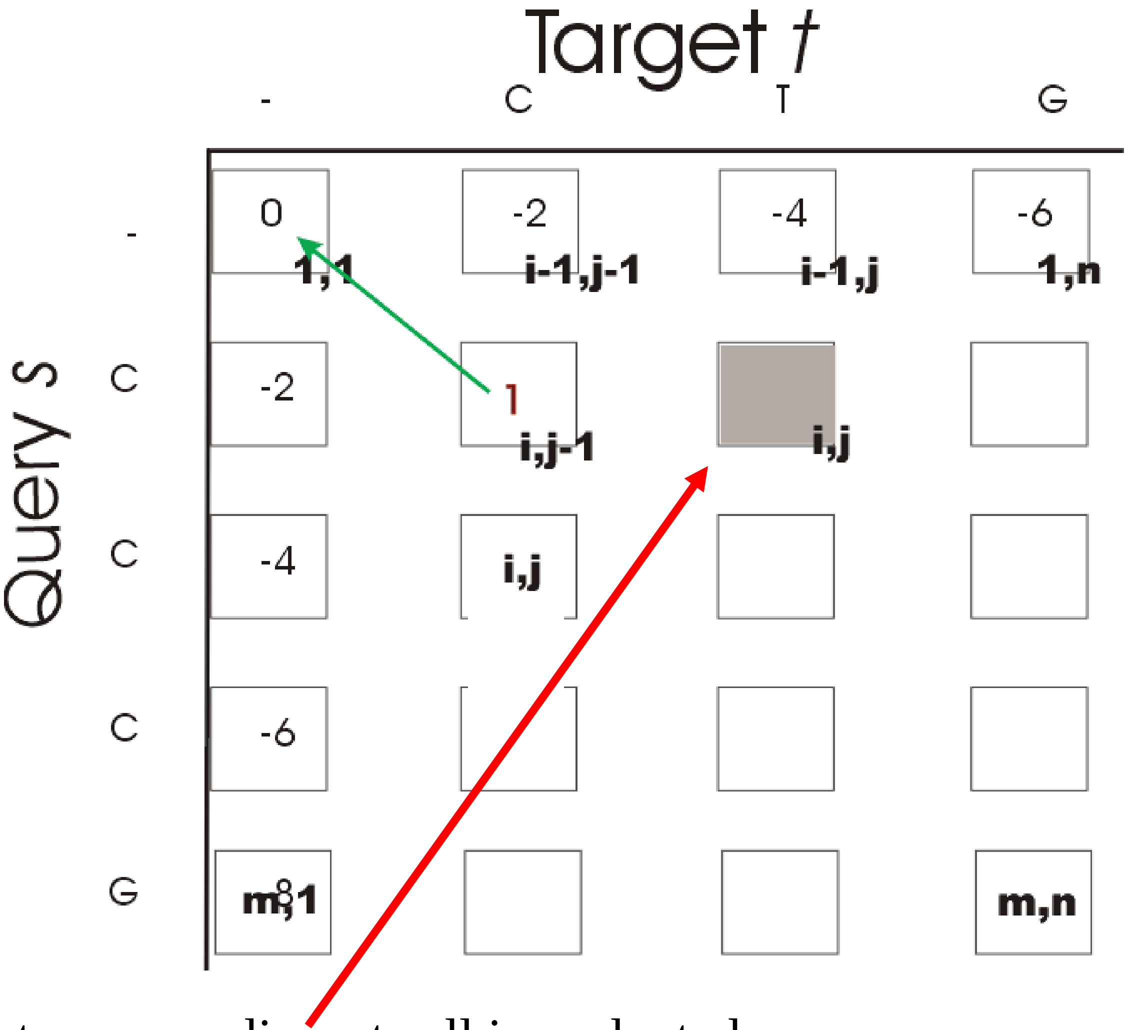
| | - | C | T | G |
|---|---|---|---|---|
| - | 0<br>**i-1,j-1** | -2<br>**i-1,j** | -4 | -6 |
| C | -2<br>**i,j-1** | **i,j** | | |
| C | -4 | | | |
| C | -6 | | | |
| G | -8 | | | **m,n** |

# Global Alignment
# Needleman-Wunsch
# Best score is selected


Target t
-
C
T
G
Query s
-
C
C
C
G
0
-2
-4
-6
-2
-4
-6
-8
i-1,j-1
i-1,j
i,j-1
i,j
m,n
Target t
-
C
Query s
-
C
0
i-1,j-1
-2
i-1,j
Add -2 for gap
Add either +1(match) or -1 (replacement) to the 0
-2
i,j-1
-2 for gap
-2-2=-4
0+1=1
-2-2=-4
i,j
C matches C so add +1 for match

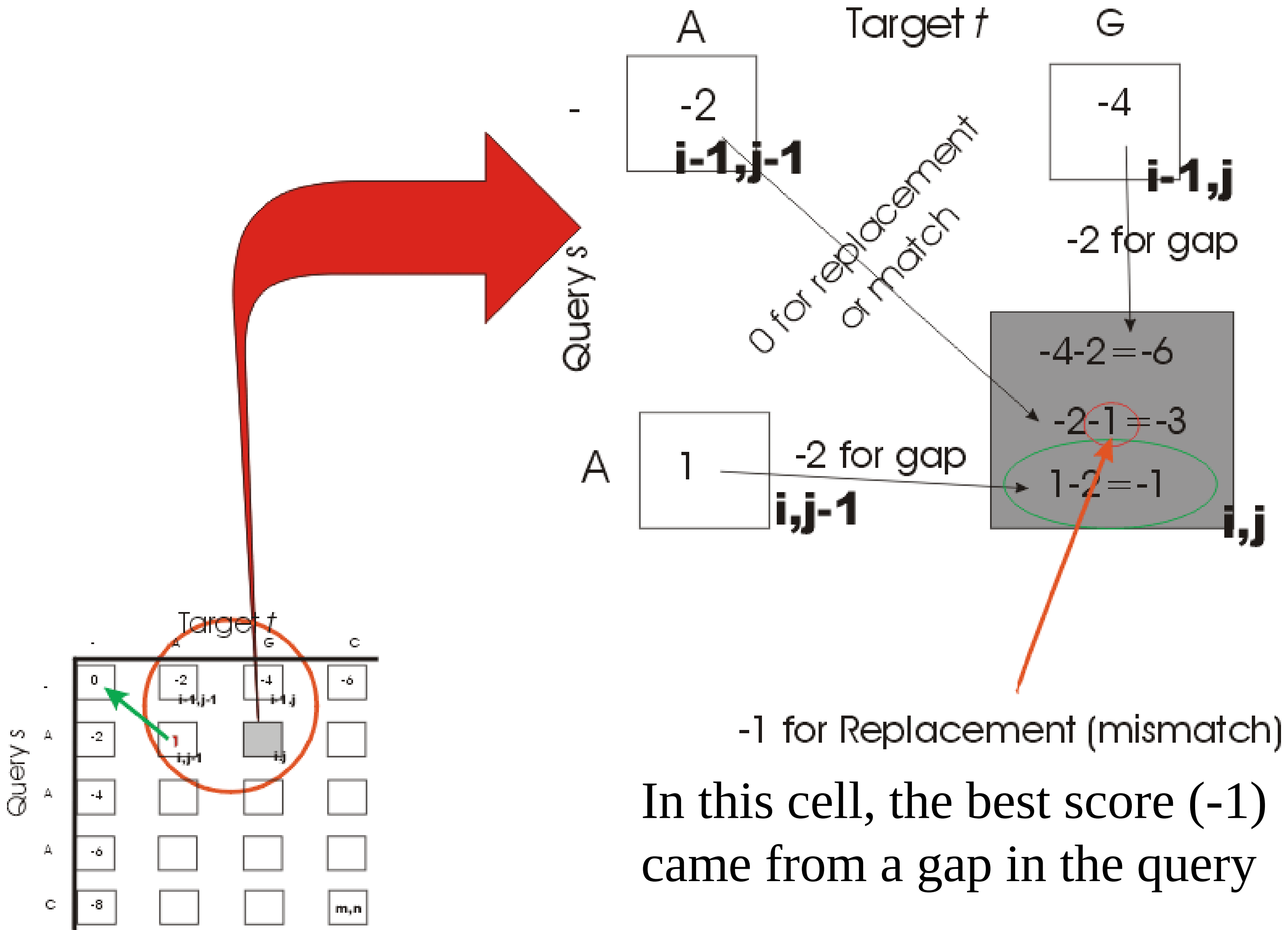
# THE TRACEBACK

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


Target t
-
C
T
G
Query s
-
C
C
C
G
0
1,1
-2
i-1,j-1
-4
i-1,j
-6
1,n
-2
1
i,j-1
i,j
-4
i,j
-6
8
m,1
m,n

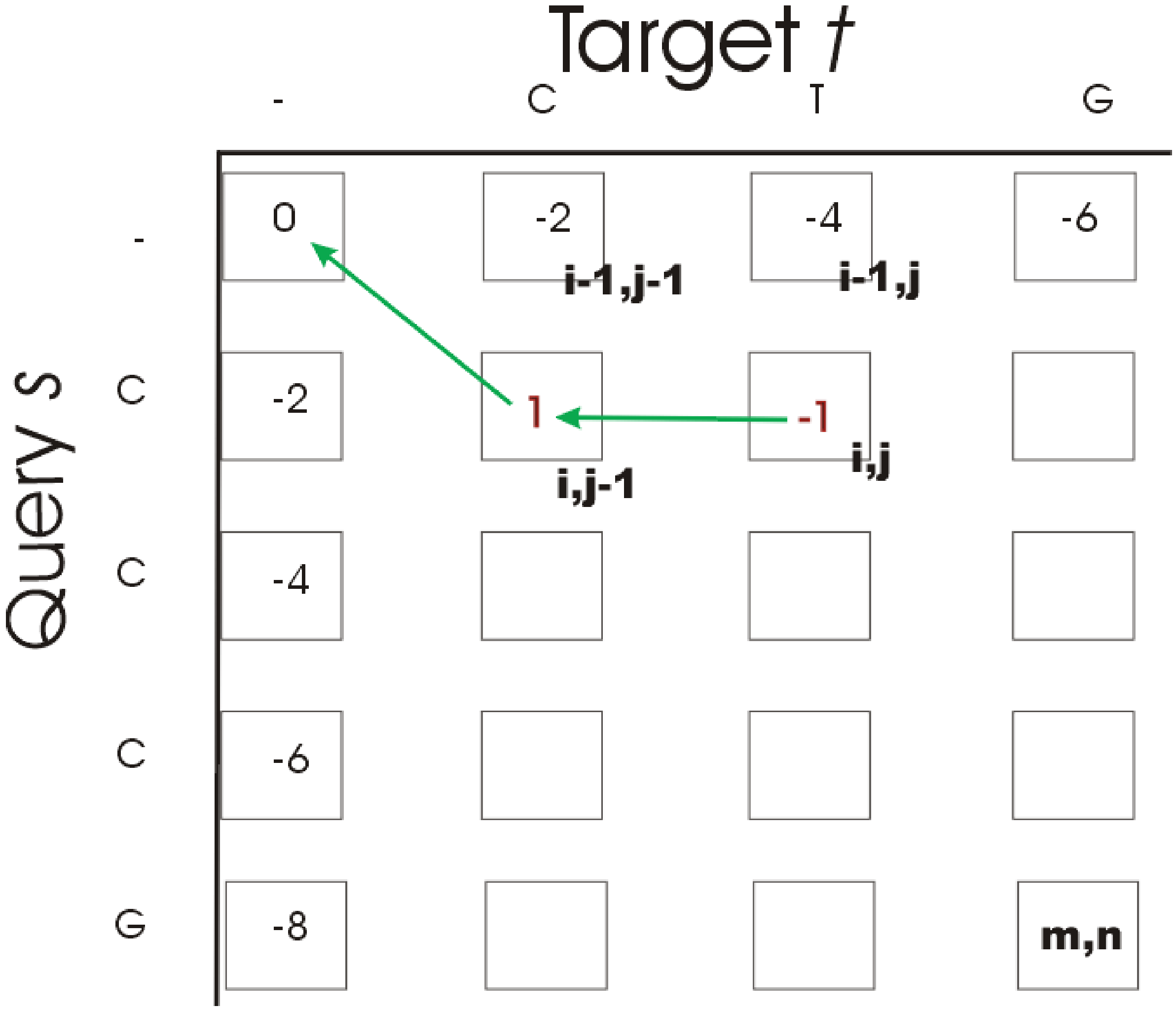
Next, a new, adjacent cell is evaluated

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


A
Target *t*
G
-
-2
i-1,j-1
-4
i-1,j
0 for replacement
or match
-2 for gap
Query s
-4-2=-6
-2-1=-3
A
1
-2 for gap
1-2=-1
i,j-1
i,j
-1 for Replacement (mismatch)
Target *t*
-
A
G
C
Query s
A
A
A
C
0
-2
-4
-6
i-1,j-1
i-1,j
-2
1
i,j-1
i,j
-4
-6
-8
m,n

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

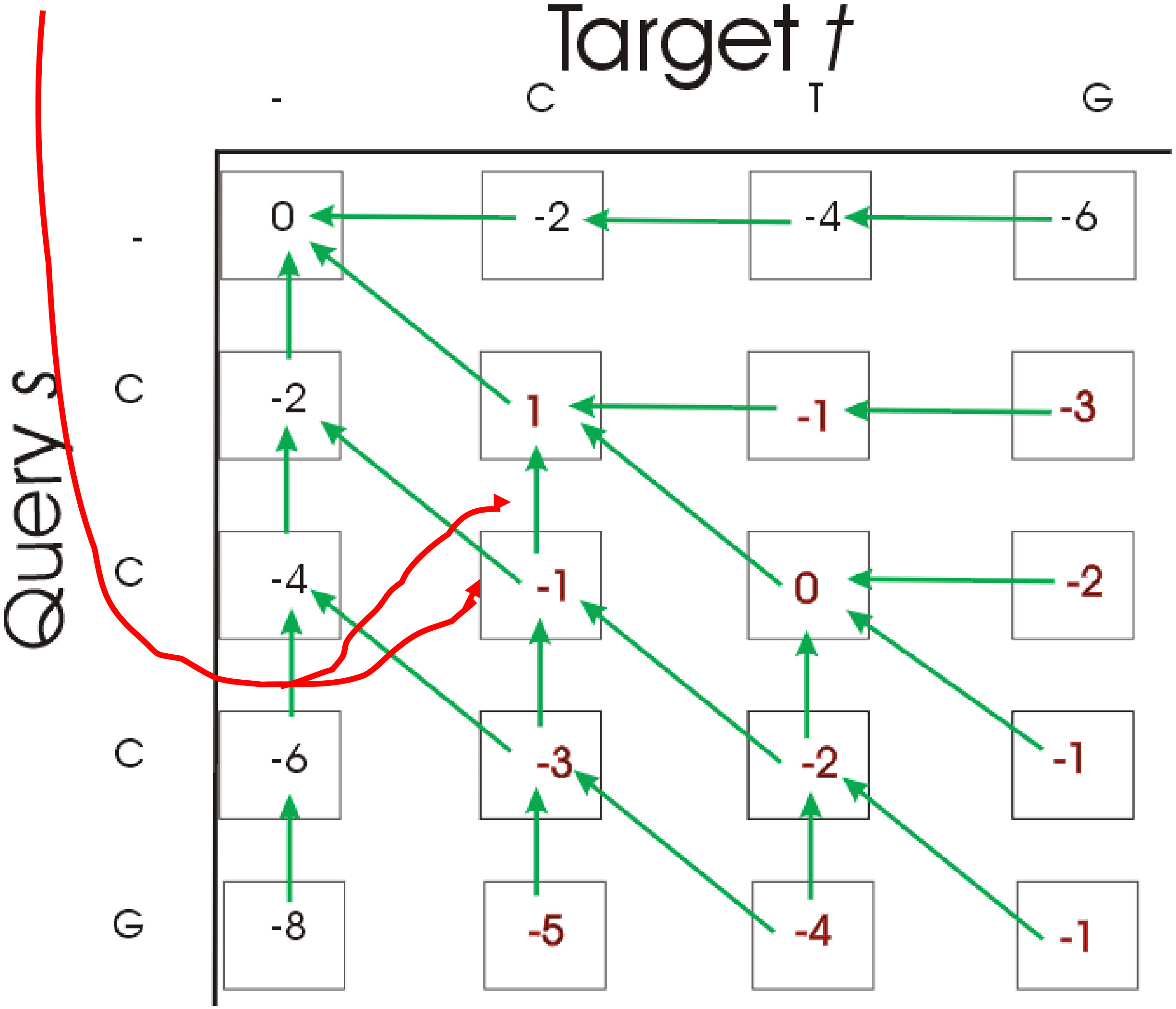

Target *t*
Query *s*
i-1,j-1
i-1,j
i,j-1
i,j

|   | - | C | T | G |
|---|---|---|---|---|
| - | 0 | -2 | -4 | -6 |
| C | -2 | 1 | -1 |  |
| C | -4 |  |  |  |
| C | -6 |  |  |  |
| G | -8 |  |  | m,n |

# 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!

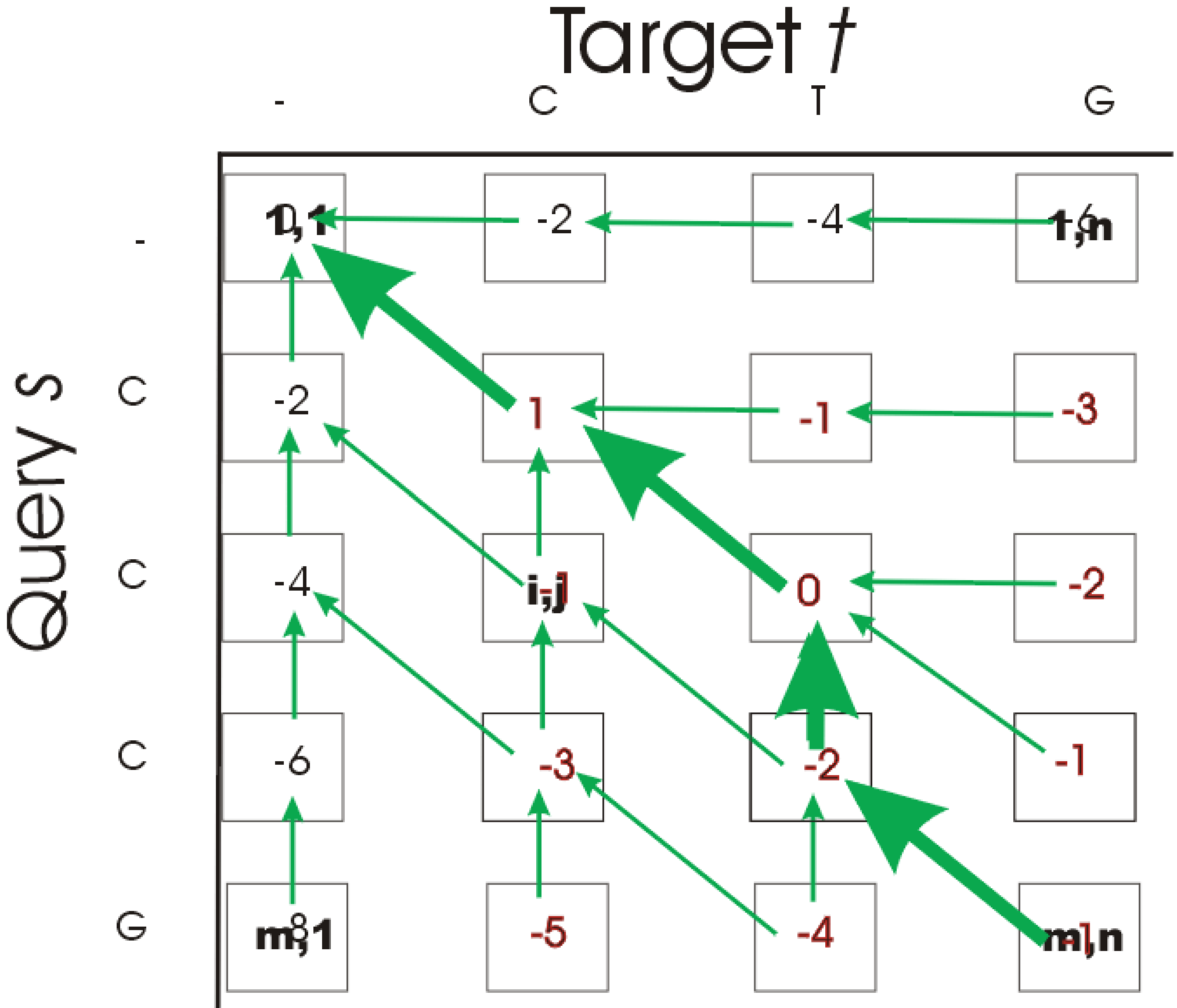


| | - | C | T | G |
|---|---|---|---|---|
| - | 0 | -2 | -4 | -6 |
| C | -2 | 1 | -1 | -3 |
| C | -4 | -1 | 0 | -2 |
| C | -6 | -3 | -2 | -1 |
| G | -8 | -5 | -4 | -1 |

# Global Alignment

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

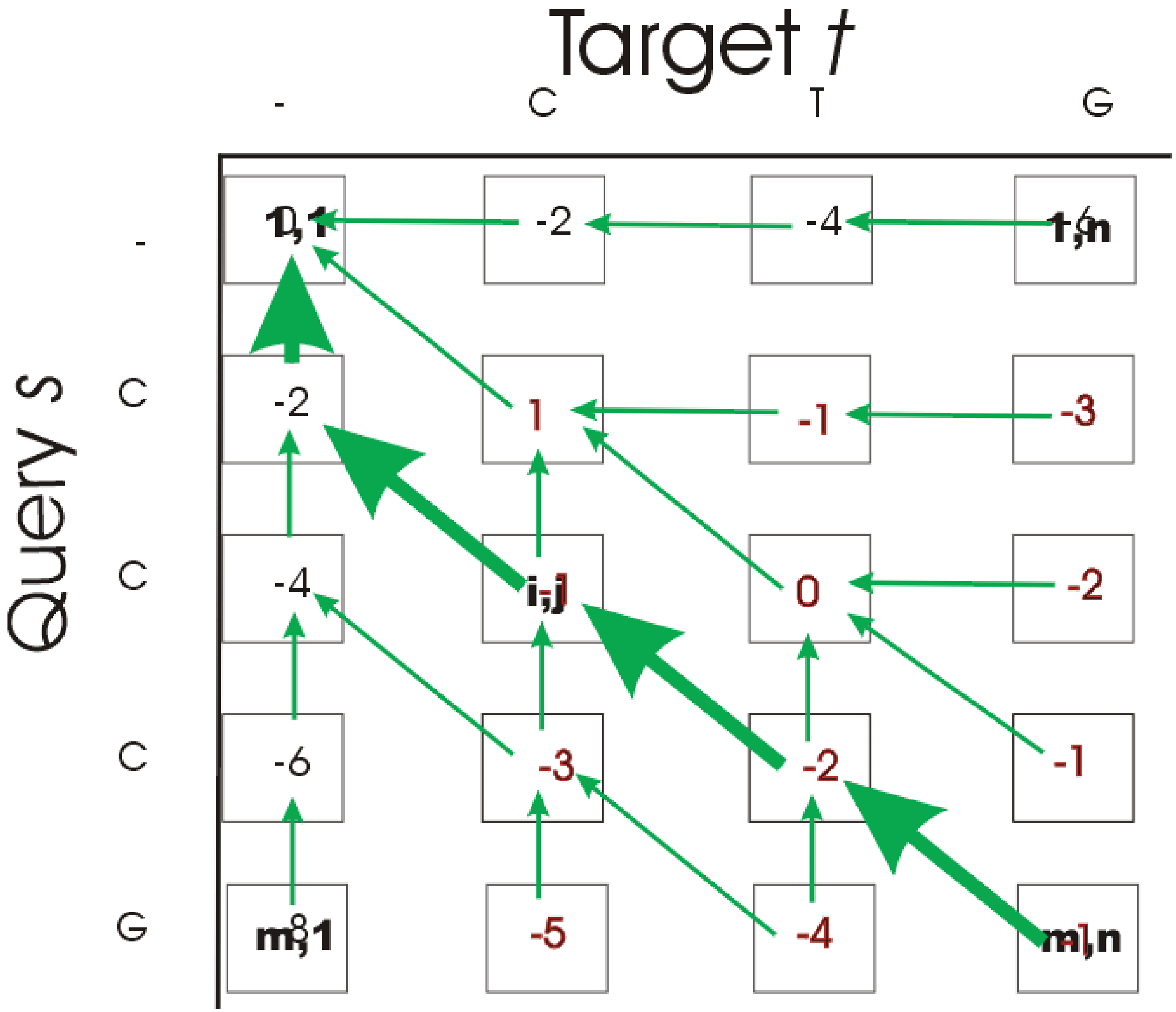
This score is -1



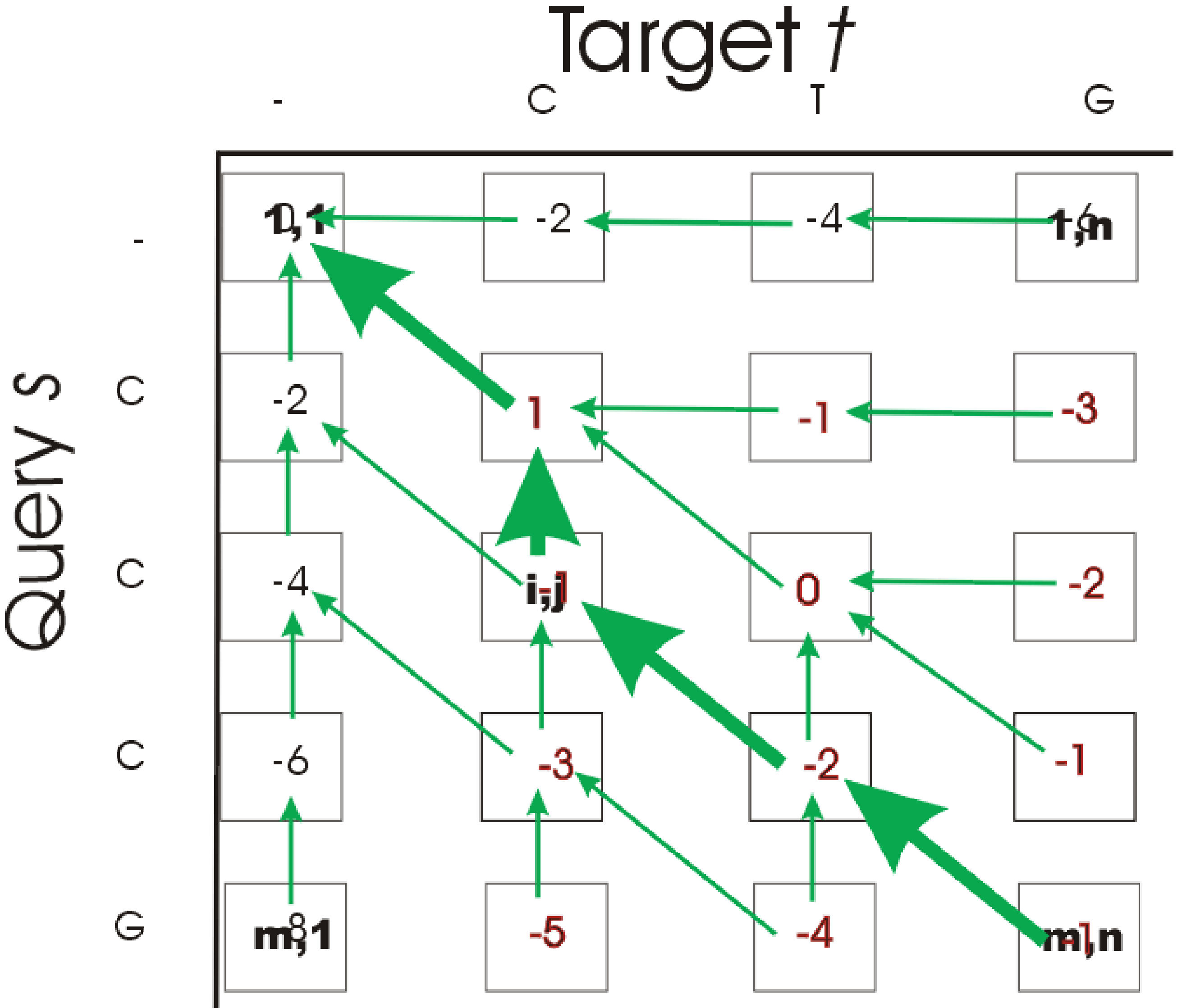
Target *t*
-
C
T
G
Query *s*
-
C
C
C
G
0
1,1
-2
-4
-6
1,n
-2
1
-1
-3
-4
i,j
0
-2
-6
-3
-2
-1
-8
m,1
-5
-4
-1
m,n

# Global Alignment

This score is –1, also


Target t
-
C
T
G
Query s
-
C
C
C
G
0
1,1
-2
-4
-6
1,n
-2
1
-1
-3
-4
i,j
-1
0
-2
-6
-3
-2
-1
-8
m,1
-5
-4
-1
m,n

# Global Alignment
This score is –1, as well



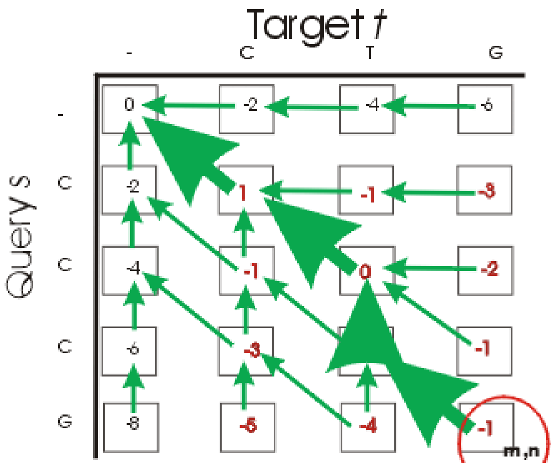
Target *t*
Query *s*

|  | - | C | T | G |
|---|---|---|---|---|
| - | 1,1<br>0 | -2 | -4 | 1,n<br>-6 |
| C | -2 | 1 | -1 | -3 |
| C | -4 | i,j<br>-1 | 0 | -2 |
| C | -6 | -3 | -2 | -1 |
| G | m,1<br>-8 | -5 | -4 | m,n<br>-1 |

# Global Alignment
# Constructing the alignment

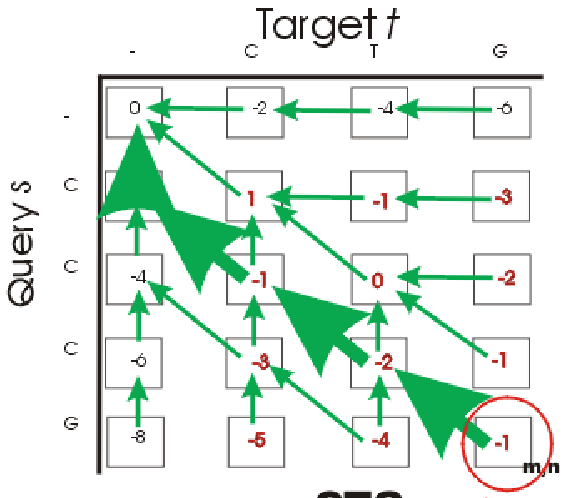
- Read off the alignment from end to start, beginning with the m,n$^{th}$ cell
  - If leaving a cell diagonally
    - Read off the query and target suffix letters (match occurred)
  - If leaving a cell horizontally
    - Read off the letter in the query
    - Read gap in the target
  - If leaving a cell vertically
    - Read off the letter in the target
    - A gap in the query

# Global Alignment


Target t
Query s
m,n

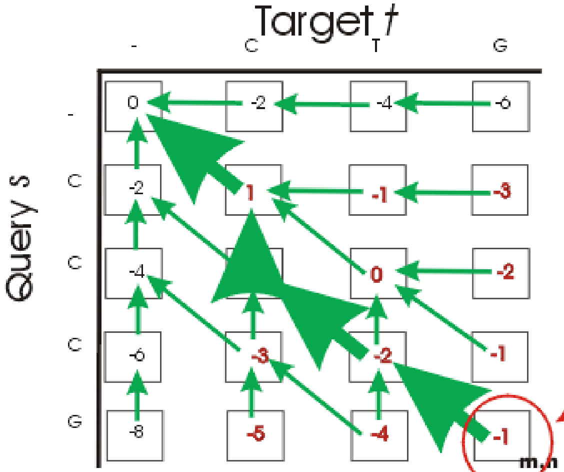
|  | - | C | T | G |
|---|---|---|---|---|
| - | 0 | -2 | -4 | -6 |
| C | -2 | 1 | -1 | -3 |
| C | -4 | -1 | 0 | -2 |
| C | -6 | -3 | [illegible] | -1 |
| G | -8 | -5 | -4 | -1 |

**CTG**


Target t
Query s
m,n

|  | - | C | T | G |
|---|---|---|---|---|
| - | 0 | -2 | -4 | -6 |
| C | [illegible] | 1 | -1 | -3 |
| C | -4 | -1 | 0 | -2 |
| C | -6 | -3 | -2 | -1 |
| G | -8 | -5 | -4 | -1 |

**-CTG**


Target t
Query s
m,n

|  | - | C | T | G |
|---|---|---|---|---|
| - | 0 | -2 | -4 | -6 |
| C | -2 | 1 | -1 | -3 |
| C | -4 | [illegible] | 0 | -2 |
| C | -6 | -3 | -2 | -1 |
| G | -8 | -5 | -4 | -1 |

**C-TG**

C T - G
CCCG
1-1-21

- C T G
CCCG
-21-11

C - T G
CCCG
1-2-11

# 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 $O((mn)^2) \cdot O(mn) = O((mn)^3)$
- 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

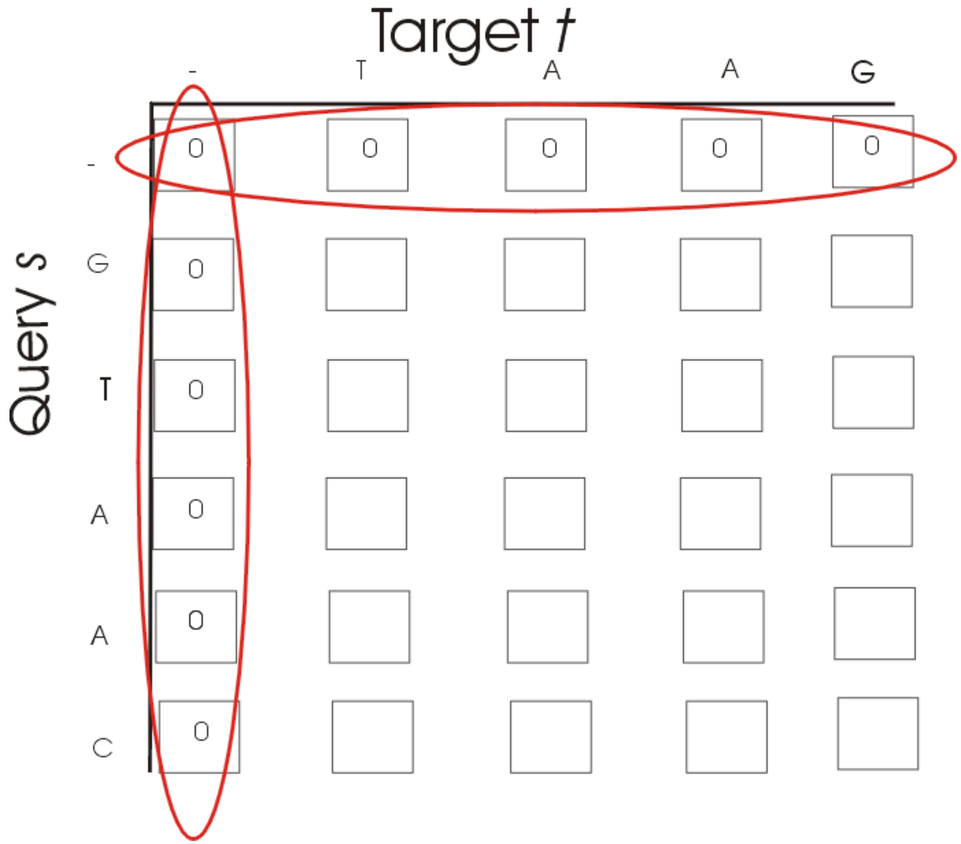
$$A(i,j) = \max \begin{cases} A(i-1,j) & -2 \quad gap \ (\text{deletion}) \\ A(i,j-1) & -2 \quad gap \ (\text{insertion}) \\ A(i-1,j-1) & \pm 1 \quad +\ \text{for match},\ -\ \text{for mismatch}\ (\text{replacement}) \\ 0 & \end{cases}$$

# 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


Target t
Query s

|  | - | T | A | A | G |
|---|---|---|---|---|---|
| - | 0 | 0 | 0 | 0 | 0 |
| G | 0 |  |  |  |  |
| T | 0 |  |  |  |  |
| A | 0 |  |  |  |  |
| A | 0 |  |  |  |  |
| C | 0 |  |  |  |  |

# Why are the first row and column all zeros?

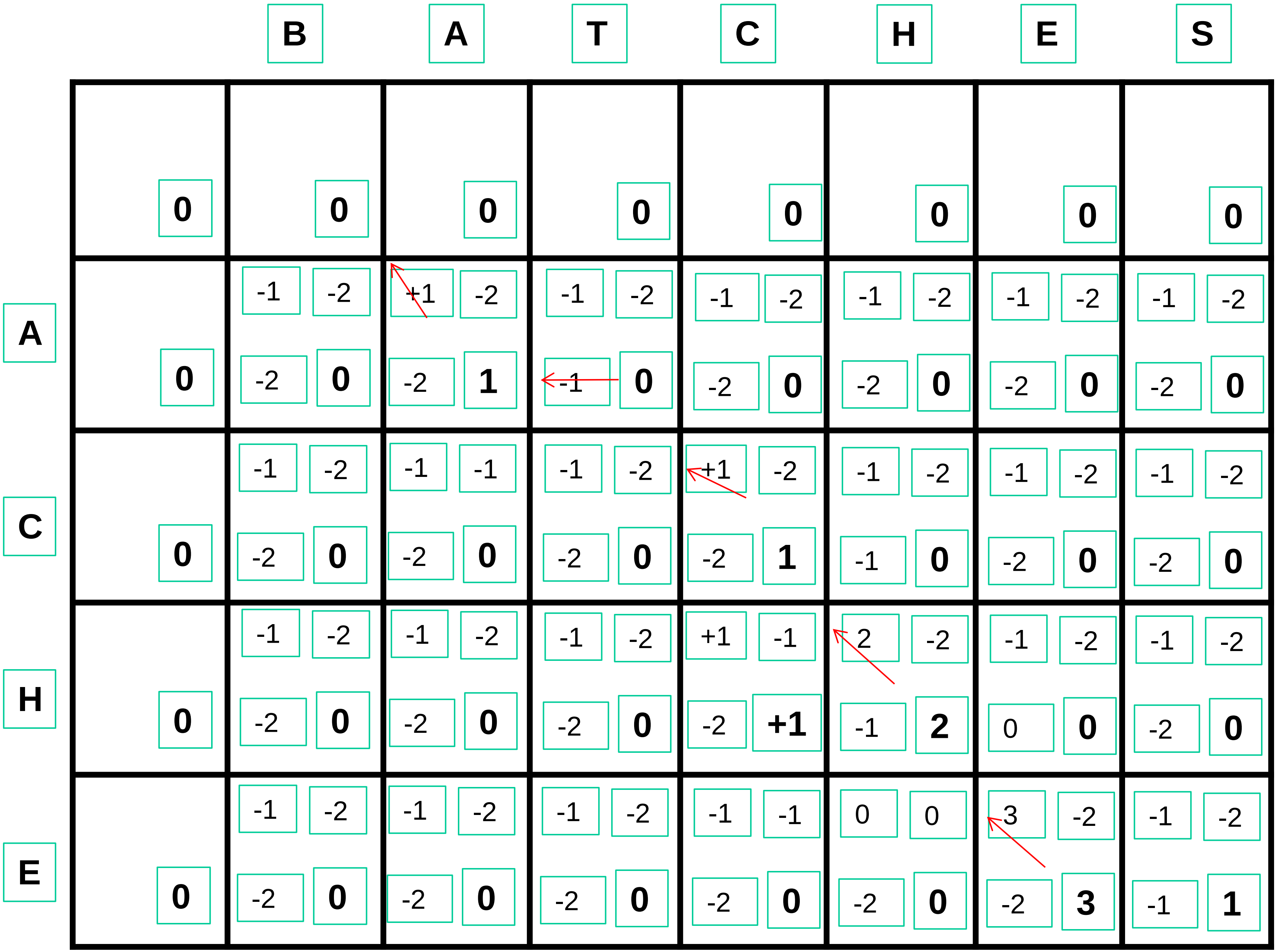
In the Needleman-Wunsch algorithm, they would be increasing negative numbers because of the inheritance penalty.

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



|  |  | B | A | T | C | H | E | S |
|---|---|---|---|---|---|---|---|---|
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 0 | -1 -2<br>-2 0 | +1 -2<br>-2 1 | -1 -2<br>-1 0 | -1 -2<br>-2 0 | -1 -2<br>-2 0 | -1 -2<br>-2 0 | -1 -2<br>-2 0 |
| C | 0 | -1 -2<br>-2 0 | -1 -1<br>-2 0 | -1 -2<br>-2 0 | +1 -2<br>-2 1 | -1 -2<br>-1 0 | -1 -2<br>-2 0 | -1 -2<br>-2 0 |
| H | 0 | -1 -2<br>-2 0 | -1 -2<br>-2 0 | -1 -2<br>-2 0 | +1 -1<br>-2 +1 | 2 -2<br>-1 2 | -1 -2<br>0 0 | -1 -2<br>-2 0 |
| E | 0 | -1 -2<br>-2 0 | -1 -2<br>-2 0 | -1 -2<br>-2 0 | -1 -1<br>-2 0 | 0 0<br>-2 0 | 3 -2<br>-2 3 | -1 -2<br>-1 1 |

# Local alignment

Reading back from the highest score, the best alignment is:

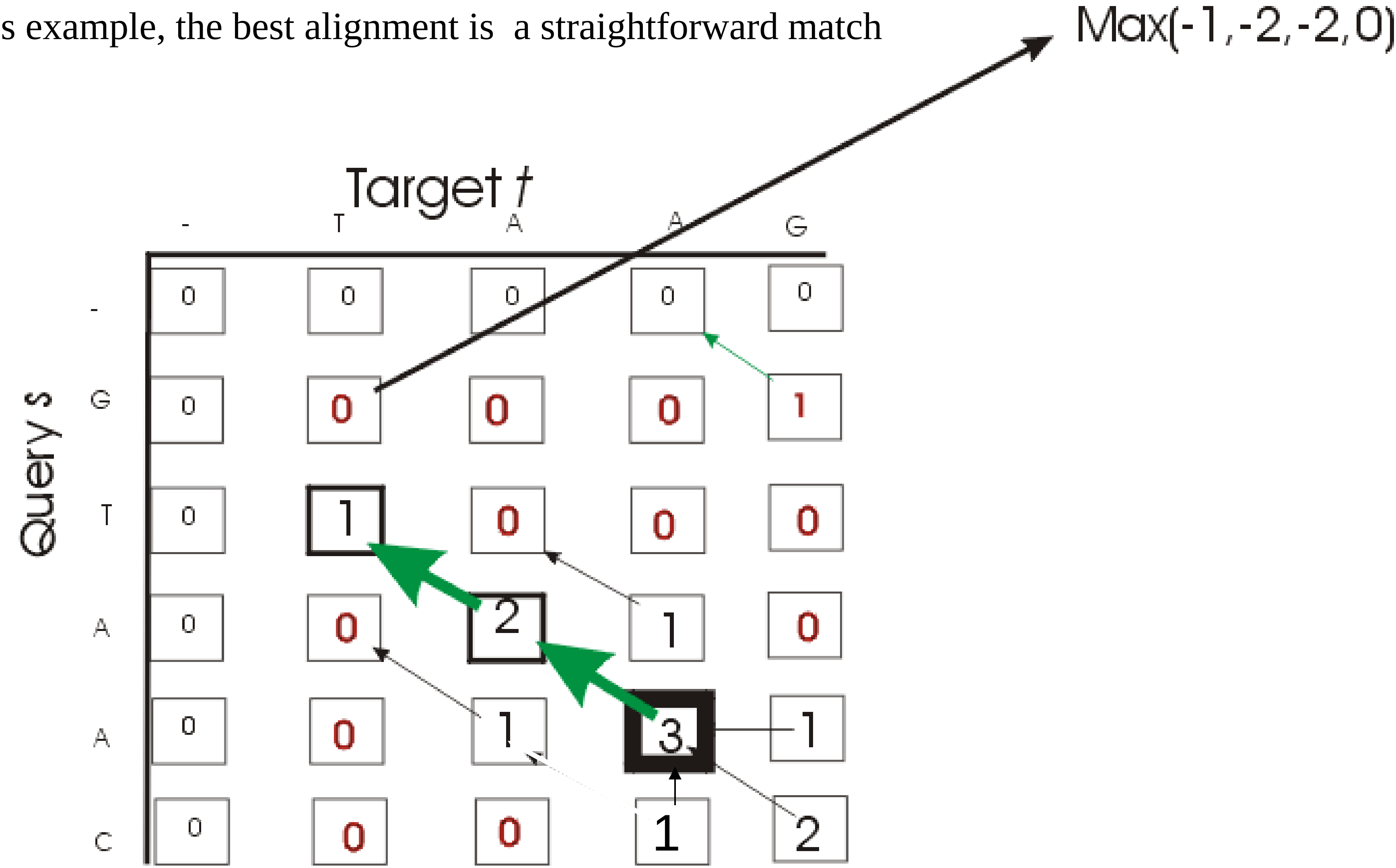
$$\begin{array}{ccccccc} B & A & T & C & H & E & S \\ & A & - & C & H & E & \end{array}$$

# 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

In this example, the best alignment is  a straightforward match


Max(-1,-2,-2,0)
Target *t*
Query *s*

| | - | T | A | A | G |
|---|---|---|---|---|---|
| - | 0 | 0 | 0 | 0 | 0 |
| G | 0 | 0 | 0 | 0 | 1 |
| T | 0 | 1 | 0 | 0 | 0 |
| A | 0 | 0 | 2 | 1 | 0 |
| A | 0 | 0 | 1 | 3 | 1 |
| C | 0 | 0 | 0 | 1 | 2 |

# Local Alignment
# Smith-Waterman Algorithm
# Results

$Highest\ scoring\ alignment$

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

Direct match!

$Lower\ scoring\ superstrings:$

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

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

# 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

$$\begin{array}{cccc} C & G & T & A \\ G & T & A & \\ A & G & A & \\ T & G & T & A \end{array} \quad \xrightarrow{\text{These 4 sequences optimally align as}} \quad \begin{array}{cccc} C & G & T & A \\ - & G & T & A \\ A & G & - & A \\ T & G & T & A \end{array}$$

# 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(1st,2nd)+Score(1st,3rd)+Score(1st,4th)

    +Score(2nd,3rd)+Score(2nd,4th)+Score(3rd,4th)
- 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 |

| | | | |
|---|---|---|---|
| *C* | *G* | *T* | *A* |
| – | *G* | *T* | *A* |
| *A* | *G* | – | *A* |
| *T* | *G* | *T* | *A* |
| –9 | 6 | –3 | 6 |

MSA Score is -15

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

$$\begin{array}{cccccc}
s_1 & C & A & T & T & T \\
s_2 & A & T & T & T & A \\
s_3 & A & T & T & G & \\
s_4 & G & C & A & T & A
\end{array}$$

# Star Alignment

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

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

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

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

# Star Alignment

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

$$\begin{array}{c|cccc} & s_1 & s_2 & s_3 & s_4 \\ \hline s_1 & \cdot & -1 & -4 & -3 \\ s_2 & -1 & \cdot & 0 & -1 \\ s_3 & -4 & 0 & \cdot & -6 \\ s_4 & -3 & -1 & -6 & \cdot \end{array}$$

# Star Alignment

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

$$\begin{array}{c|ccccc}
 & s_1 & s_2 & s_3 & s_4 & \\
\hline
s_1 & & -1 & -4 & -3 & = -8 \\
s_2 & -1 & & 0 & -1 & = -2 \\
s_3 & -4 & 0 & & -6 & = -10 \\
s_4 & -3 & -1 & -6 & & = -10
\end{array}$$

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

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

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

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

# Final Product of the Star Alignment

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

$$\begin{array}{cc|c|c|c|c|c}
s_2 & - & A & T & T & T & A \\
s_1 & C & A & T & T & T & - \\
s_3 & - & A & T & T & G & - \\
s_4 & - & G & C & A & T & A
\end{array}$$

# 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 $\mathcal{O}(kn+kl)$