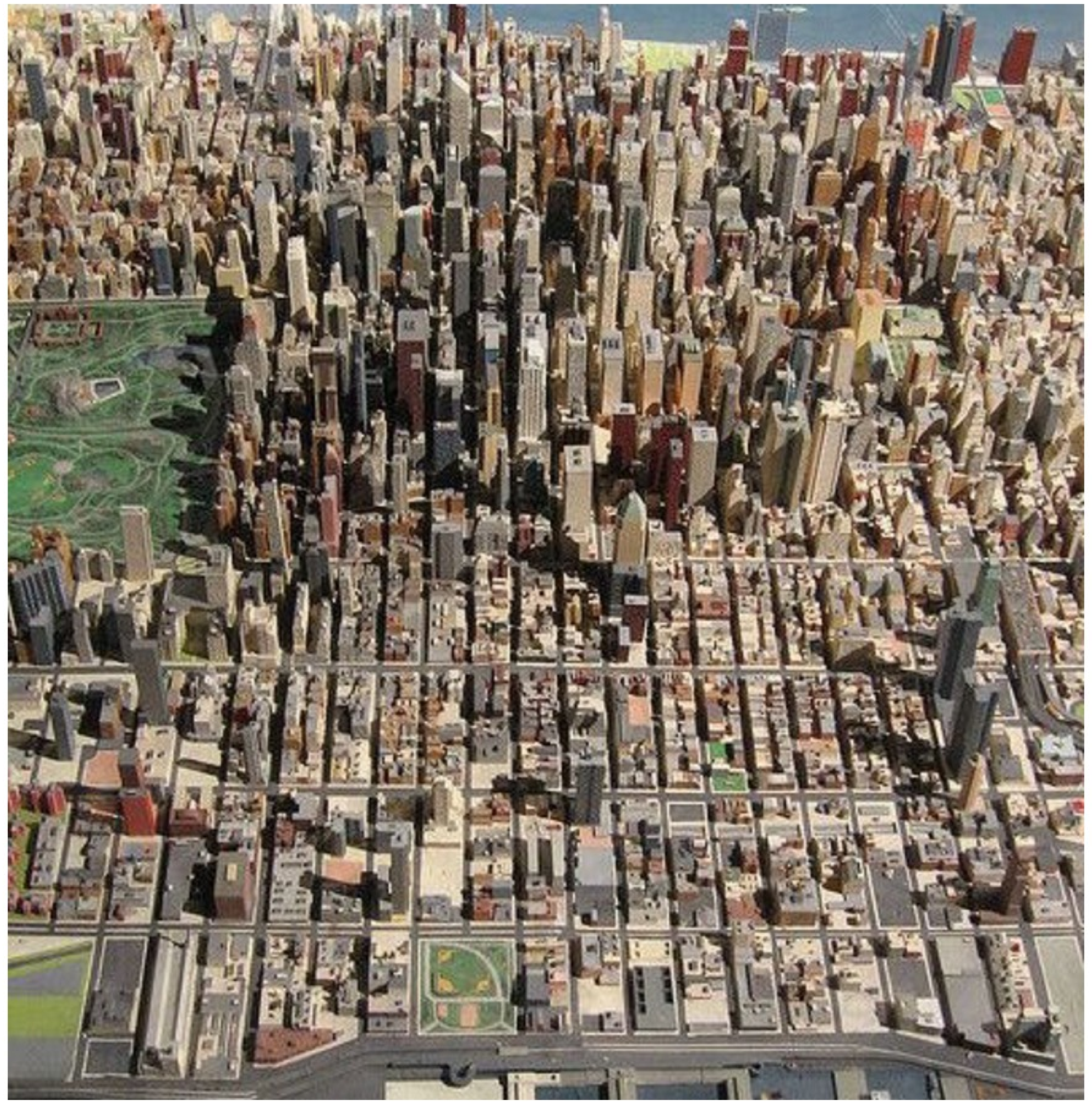
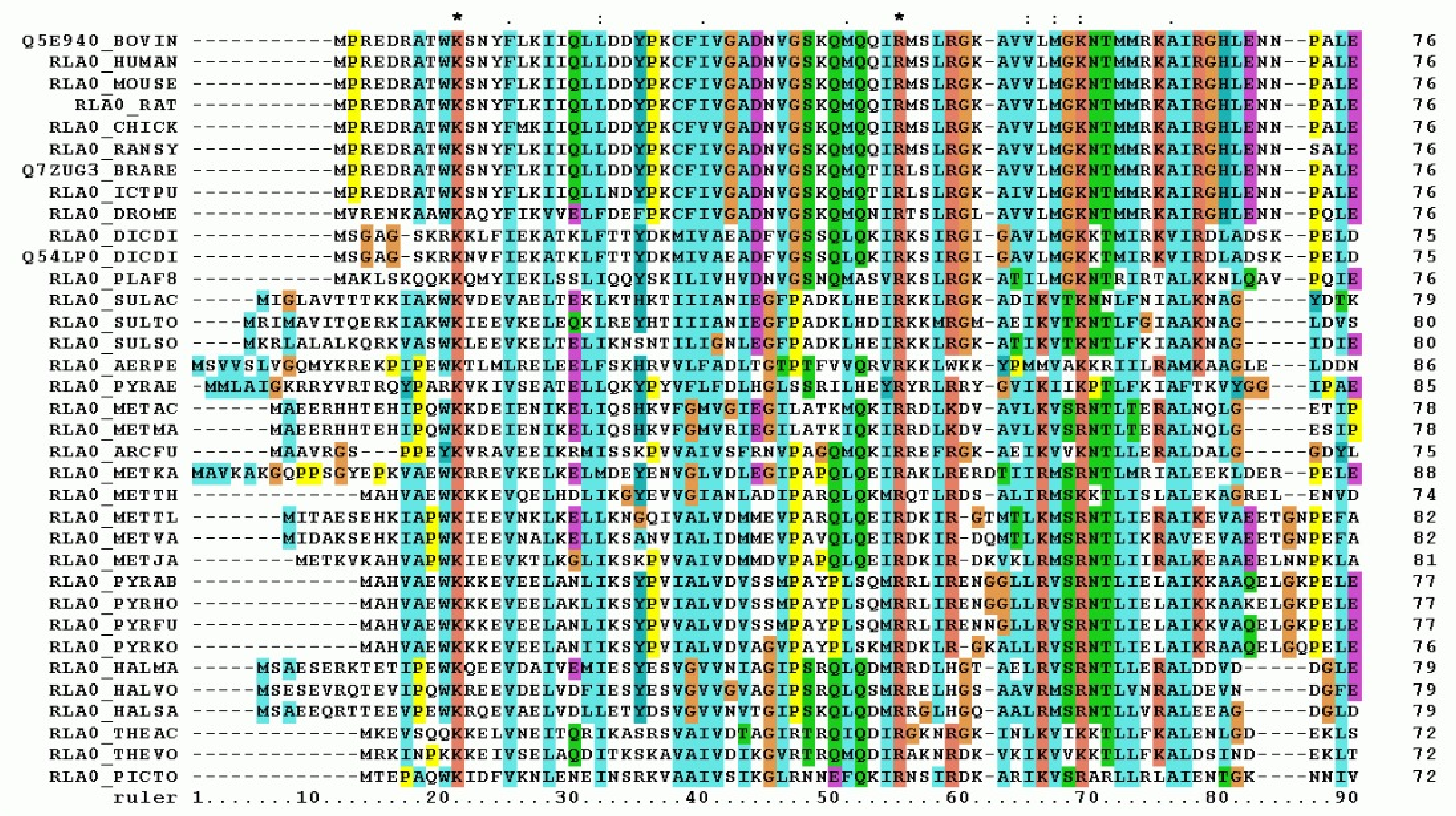
# Sequence Alignment



- Relating sequence alignment to our Manhattan Tour Problem
- Midterm exam (a week from Wednesday 3/8) covers up to this lecture

# A Biological Dynamic Programming Problem

- How to measure the similarity between a pair of nucleotide or amino acid sequences
- When Motif-Searching Problem we used Hamming distance as our measure
- Is Hamming distance the best measure?
- How can we distinguish matches that occur by chance from slightly modified patterns?
- What sorts of modifications should we allow?



```
                                 *   .      :       .          .   *         : : :      .
Q5E940_BOVIN -----------MPREDRATWKSNYFLKIIQLLDDYPKCFIVGADNVGSKQMQQIRMSLRGK-AVVLMGKNTMMRKAIRGHLENN--PALE   76
  RLA0_HUMAN -----------MPREDRATWKSNYFLKIIQLLDDYPKCFIVGADNVGSKQMQQIRMSLRGK-AVVLMGKNTMMRKAIRGHLENN--PALE   76
  RLA0_MOUSE -----------MPREDRATWKSNYFLKIIQLLDDYPKCFIVGADNVGSKQMQQIRMSLRGK-AVVLMGKNTMMRKAIRGHLENN--PALE   76
    RLA0_RAT -----------MPREDRATWKSNYFLKIIQLLDDYPKCFIVGADNVGSKQMQQIRMSLRGK-AVVLMGKNTMMRKAIRGHLENN--PALE   76
  RLA0_CHICK -----------MPREDRATWKSNYFMKIIQLLDDYPKCFVVGADNVGSKQMQQIRMSLRGK-AVVLMGKNTMMRKAIRGHLENN--PALE   76
  RLA0_RANSY -----------MPREDRATWKSNYFLKIIQLLDDYPKCFIVGADNVGSKQMQQIRMSLRGK-AVVLMGKNTMMRKAIRGHLENN--SALE   76
Q7ZUG3_BRARE -----------MPREDRATWKSNYFLKIIQLLDDYPKCFIVGADNVGSKQMQTIRLSLRGK-AVVLMGKNTMMRKAIRGHLENN--PALE   76
  RLA0_ICTPU -----------MPREDRATWKSNYFLKIIQLLNDYPKCFIVGADNVGSKQMQTIRLSLRGK-AIVLMGKNTMMRKAIRGHLENN--PALE   76
  RLA0_DROME -----------MVRENKAAWKAQYFIKVVELFDEFPKCFIVGADNVGSKQMQNIRTSLRGL-AVVLMGKNTMMRKAIRGHLENN--PQLE   76
  RLA0_DICDI -----------MSGAG-SKRKKLFIEKATKLFTTYDKMIVAEADFVGSSQLQKIRKSIRGI-GAVLMGKKTMIRKVIRDLADSK--PELD   75
Q54LP0_DICDI -----------MSGAG-SKRKNVFIEKATKLFTTYDKMIVAEADFVGSSQLQKIRKSIRGI-GAVLMGKKTMIRKVIRDLADSK--PELD   75
  RLA0_PLAF8 -----------MAKLSKQQKKQMYIEKLSSLIQQYSKILIVHVDNVGSNQMASVRKSLRGK-ATILMGKNTRIRTALKKNLQAV--PQIE   76
  RLA0_SULAC -----MIGLAVTTTKKIAKWKVDEVAELTEKLKTHKTIIIANIEGFPADKLHEIRKKLRGK-ADIKVTKNNLFNIALKNAG-----YDTK   79
  RLA0_SULTO ----MRIMAVITQERKIAKWKIEEVKELEQKLREYHTIIIANIEGFPADKLHDIRKKMRGM-AEIKVTKNTLFGIAAKNAG-----LDVS   80
  RLA0_SULSO ----MKRLALALKQRKVASWKLEEVKELTELIKNSNTILIGNLEGFPADKLHEIRKKLRGK-ATIKVTKNTLFKIAAKNAG-----IDIE   80
  RLA0_AERPE MSVVSLVGQMYKREKPIPEWKTLMLRELEELFSKHRVVLFADLTGTPTFVVQRVRKKLWKK-YPMMVAKKRIILRAMKAAGLE---LDDN   86
  RLA0_PYRAE -MMLAIGKRRYVRTRQYPARKVKIVSEATELLQKYPYVFLFDLHGLSSRILHEYRYRLRRY-GVIKIIKPTLFKIAFTKVYGG---IPAE   85
  RLA0_METAC ------MAEERHHTEHIPQWKKDEIENIKELIQSHKVFGMVGIEGILATKMQKIRRDLKDV-AVLKVSRNTLTERALNQLG-----ETIP   78
  RLA0_METMA ------MAEERHHTEHIPQWKKDEIENIKELIQSHKVFGMVRIEGILATKIQKIRRDLKDV-AVLKVSRNTLTERALNQLG-----ESIP   78
  RLA0_ARCFU ------MAAVRGS---PPEYKVRAVEEIKRMISSKPVVAIVSFRNVPAGQMQKIRREFRGK-AEIKVVKNTLLERALDALG-----GDYL   75
  RLA0_METKA MAVKAKGQPPSGYEPKVAEWKRREVKELKELMDEYENVGLVDLEGIPAPQLQEIRAKLRERDTIIRMSRNTLMRIALEEKLDER--PELE   88
  RLA0_METTH -------------MAHVAEWKKKEVQELHDLIKGYEVVGIANLADIPARQLQKMRQTLRDS-ALIRMSKKTLISLALEKAGREL--ENVD   74
  RLA0_METTL -------MITAESEHKIAPWKIEEVNKLKELLKNGQIVALVDMMEVPARQLQEIRDKIR-GTMTLKMSRNTLIERAIKEVAEETGNPEFA   82
  RLA0_METVA -------MIDAKSEHKIAPWKIEEVNALKELLKSANVIALIDMMEVPAVQLQEIRDKIR-DQMTLKMSRNTLIKRAVEEVAEETGNPEFA   82
  RLA0_METJA --------METKVKAHVAPWKIEEVKTLKGLIKSKPVVAIVDMMDVPAPQLQEIRDKIR-DKVKLRMSRNTLIIRALKEAAEELNNPKLA   81
  RLA0_PYRAB -------------MAHVAEWKKKEVEELANLIKSYPVIALVDVSSMPAYPLSQMRRLIRENGGLLRVSRNTLIELAIKKAAQELGKPELE   77
  RLA0_PYRHO -------------MAHVAEWKKKEVEELAKLIKSYPVIALVDVSSMPAYPLSQMRRLIRENGGLLRVSRNTLIELAIKKAAKELGKPELE   77
  RLA0_PYRFU -------------MAHVAEWKKKEVEELANLIKSYPVVALVDVSSMPAYPLSQMRRLIRENNGLLRVSRNTLIELAIKKVAQELGKPELE   77
  RLA0_PYRKO -------------MAHVAEWKKKEVEELANIIKSYPVIALVDVAGVPAYPLSKMRDKLR-GKALLRVSRNTLIELAIKRAAQELGQPELE   76
  RLA0_HALMA -----MSAESERKTETIPEWKQEEVDAIVEMIESYESVGVVNIAGIPSRQLQDMRRDLHGT-AELRVSRNTLLERALDDVD-----DGLE   79
  RLA0_HALVO -----MSESEVRQTEVIPQWKREEVDELVDFIESYESVGVVGVAGIPSRQLQSMRRELHGS-AAVRMSRNTLVNRALDEVN-----DGFE   79
  RLA0_HALSA -----MSAEEQRTTEEVPEWKRQEVAELVDLLETYDSVGVVNVTGIPSKQLQDMRRGLHGQ-AALRMSRNTLLVRALEEAG-----DGLD   79
  RLA0_THEAC -------------MKEVSQQKKELVNEITQRIKASRSVAIVDTAGIRTRQIQDIRGKNRGK-INLKVIKKTLLFKALENLGD----EKLS   72
  RLA0_THEVO -------------MRKINPKKKEIVSELAQDITKSKAVAIVDIKGVRTRQMQDIRAKNRDK-VKIKVVKKTLLFKALDSIND----EKLT   72
  RLA0_PICTO -------------MTEPAQWKIDFVKNLENEINSRKVAAIVSIKGLRNNEFQKIRNSIRDK-ARIKVSRARLLRLAIENTGK----NNIV   72
       ruler 1.......10........20........30........40........50........60........70........80........90
```

# Best Sequence Matches

- Depends on how you define *Best*
- Consider the two DNA sequences $v$ and $w$:

```
u:  TAGACAT_
v:  AGAGACAT
    11111111
```

- The Hamming distance: $dH(v, w) = 8$ is large but the sequences are very similar
- What if we allowed insertions and deletions?

# Allowing Insertions and Deletions

- By shifting one sequence over one position:

```
u: _TAGACAT
v: AGAGACAT
   11000000
```

- The edit distance: *dH(v, w) = 2*.
- Hamming distance neglects insertions and deletions

First Previous Next Notebook

# Edit Distance

- Levenshtein introduced the notion of an “edit distance” between two strings as the minimum number of elementary operations (insertions, deletions, and substitutions) to transform one string into the other in 1965.
- d(v,w) = Minimum number of elementary operations to transform $v \rightarrow w$
- Computing Hamming distance is a trivial task
- Computing edit distance is less trivial

# Edit Distance: Example

```
TGCATAT → ATCCGAT in 5 steps

TGCATAT → (DELETE last T)
TGCATA  → (DELETE last A)
TGCAT   → (INSERT A at front)
ATGCAT  → (SUBSTITUTE C for G)
ATCCAT  → (INSERT G before last A)
ATCCGAT   (Done)
```

What is the edit distance? 5?

# Edit Distance: Example (2nd Try)

```
TGCATAT → ATCCGAT in 4 steps

TGCATAT  → (INSERT A at front)
ATGCATAT → (DELETE 2nd T)
ATGCAAT  → (SUBSTITUTE G for 2nd A)
ATGCGAT  → (SUBSTITUTE C for 1st G)
ATCCGAT    (Done)
```

Is 4 the minimum edit distance? 3?

- A little jargon: Since the effect of insertion in one string can be accomplished via a deletion in the other string these two operations are quite similar. Often algorithms will consider them together as a single operation called INDEL

# Longest Common Subsequence

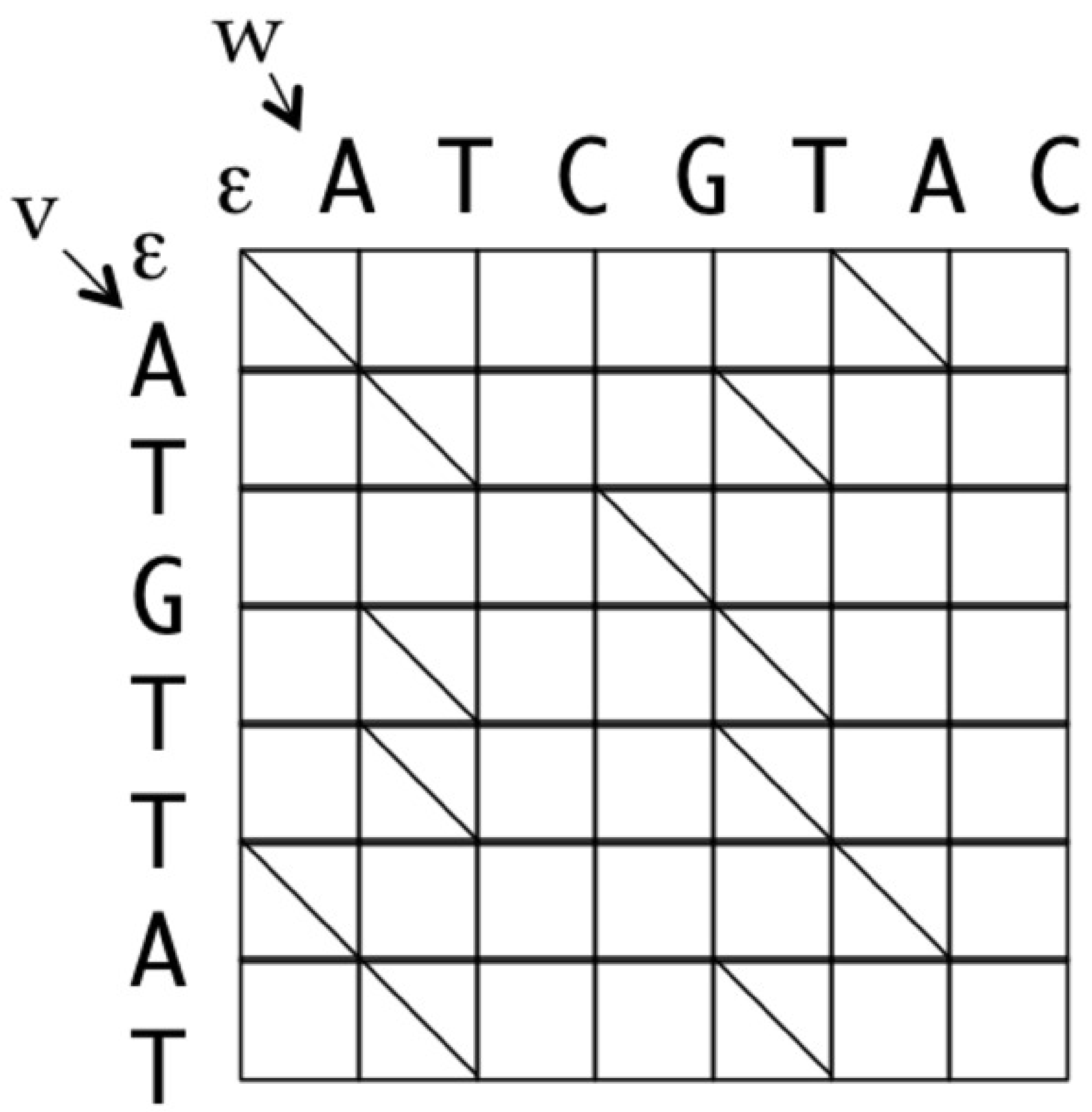
- A special case of edit distance where no *substitutions* are allowed
- A subsequence need not be contiguous, but the symbol order must be preserved
  Ex. If v = ATTGCTA then AGCA and TTTA are subsequences of v, but TGTT and ACGA are not
- All substrings of *v* are subsequences, but not vice versa
- The edit distance, *d*, is related to the length of the LCS, *s*, by:

$$d(u, w) = len(v) + len(w) – 2s(u, w)$$

```
ANUNCLEIKE
UNCBEATDUKE

anUNC_lE____iKE  10 - 6 = 4
__UNCb_Eatdu_KE  11 - 6 = 5
```

# LCS as a Dynamic Program



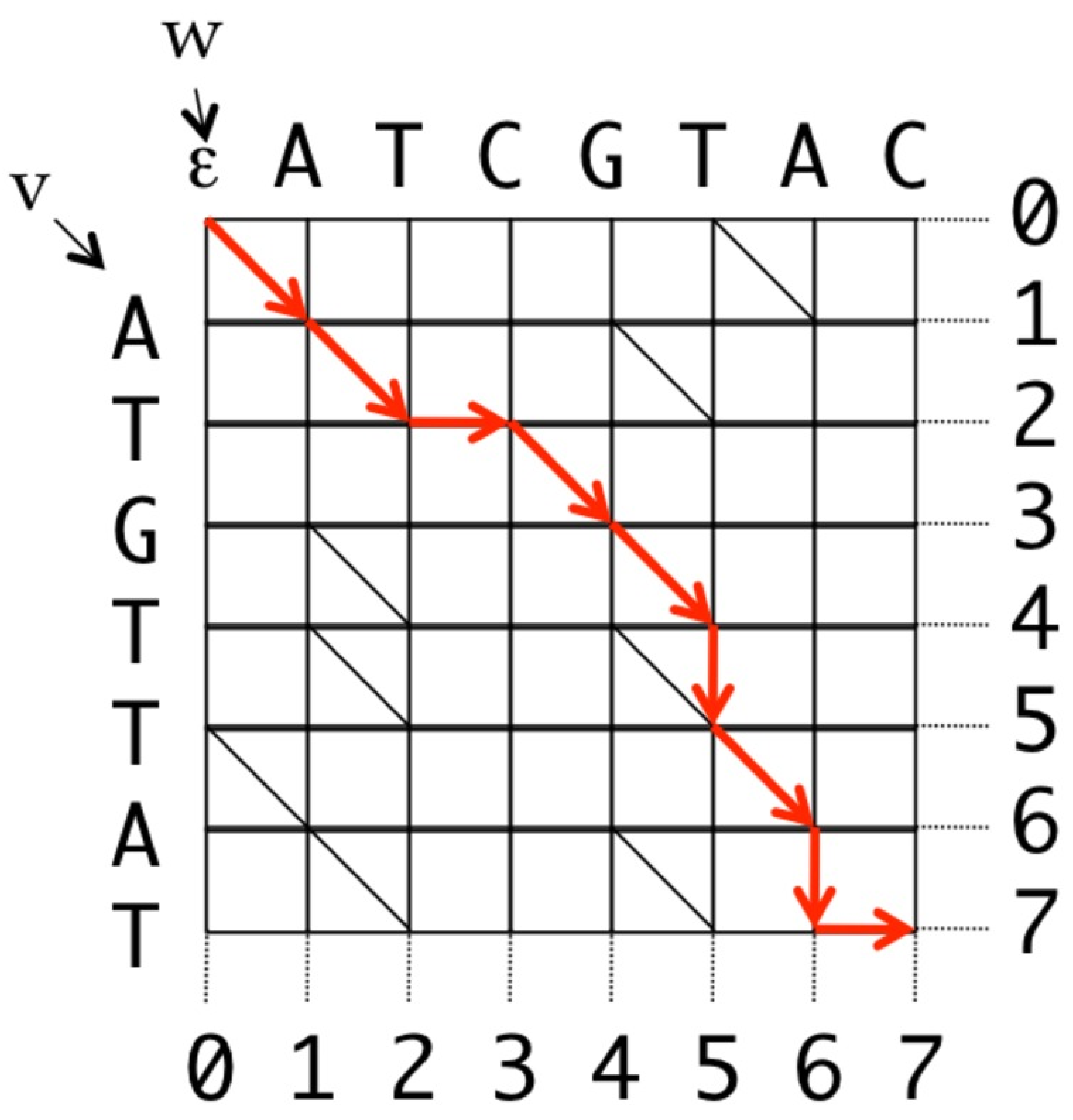
- All possible possible alignments can be represented as a path from the string's beginning (source) to its end (destination)
- Horizontal edges add gaps in v. Vertical edges add gaps in w. Diagonal edges are a match
- Notice that we've only included valid diagonal edges appear in our graph

# Various Alignments

- Introduce coordinates for the grid
- All valid paths from the source to the destination represent some alignment

```
0 1 2 2 3 4 5 6 7 7
v A T _ G T T A T _
w A T C G T _ A _ C
0 1 2 3 4 5 5 6 6 7
```

- Path: (0,0), (1,1), (2,2), (2,3), (3,4), (4,5), (5,5), (6,6), (7,6), (7,7)

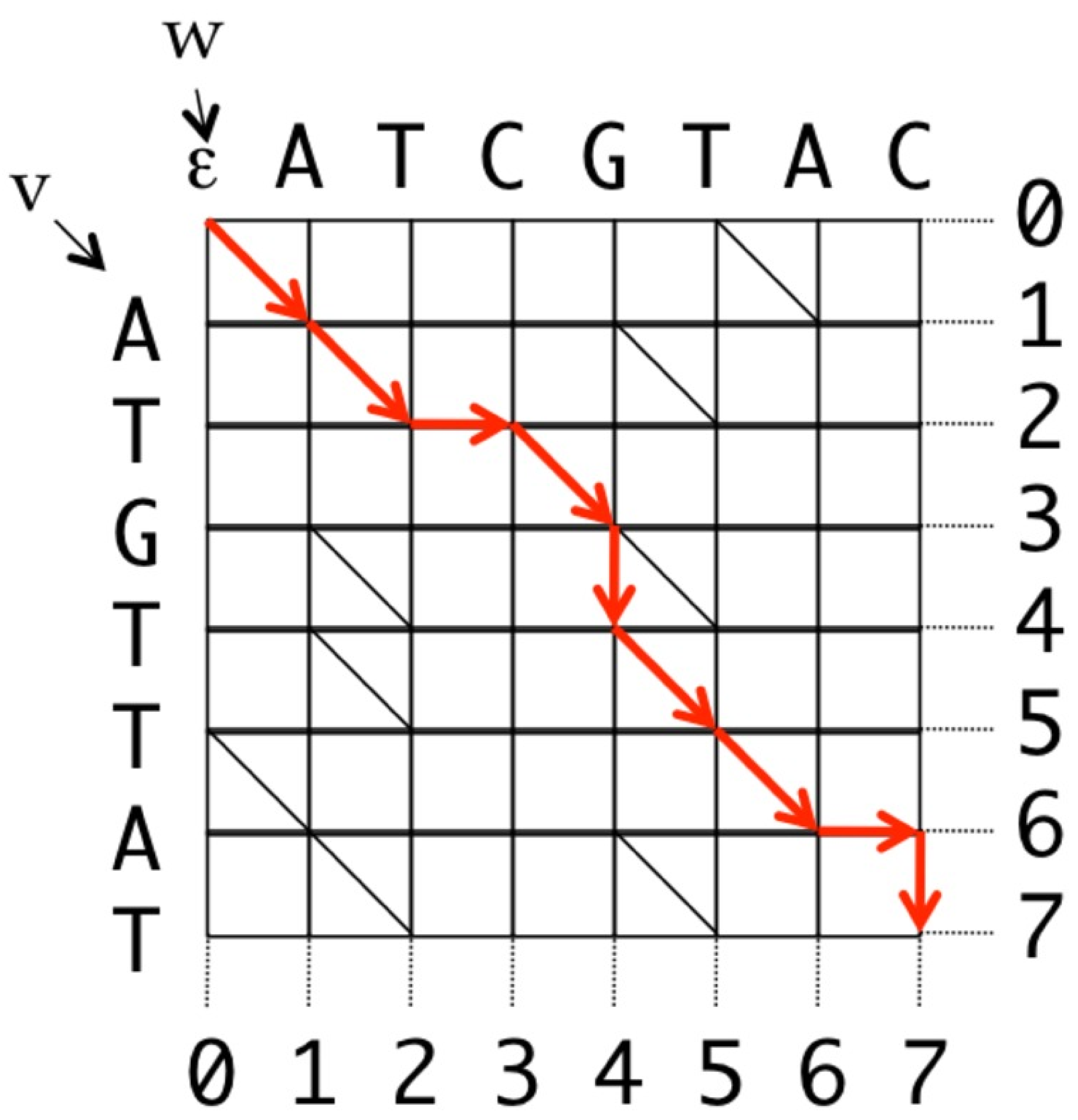


# Alternate Alignment

- Introduce coordinates for the grid
- All valid paths from the source to the destination represent some alignment

```
0 1 2 2 3 4 5 6 6 7
v A T _ G T T A _ T
w A T C G _ T A C _
0 1 2 3 4 4 5 6 7 7
```

- Path: (0,0), (1,1), (2,2), (2,3), (3,4), (4,4), (5,5), (6,6), (6,7), (7,7)

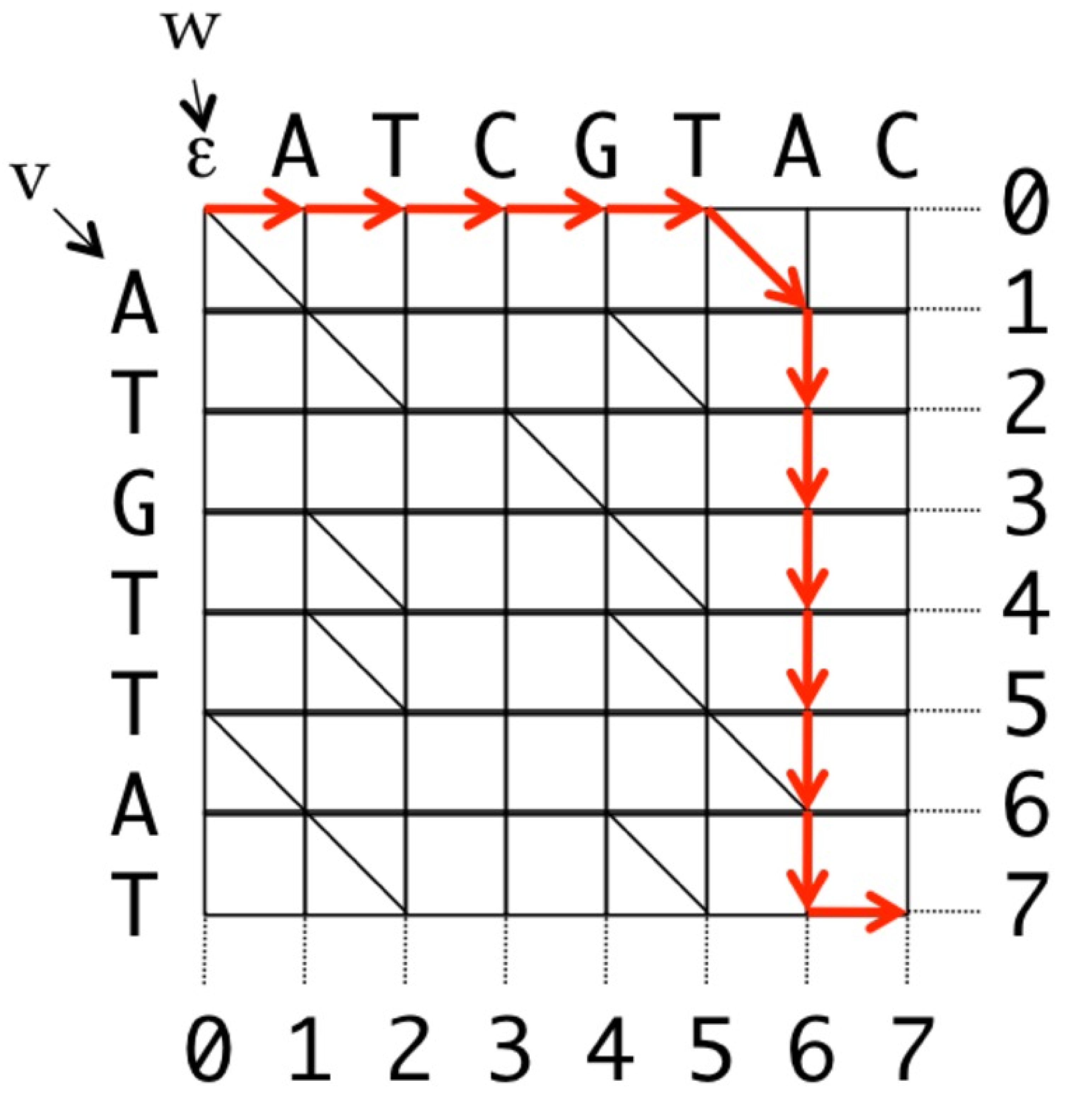


# Even Bad Alignments

- Introduce coordinates for the grid
- All valid paths from the source to the destination represent some alignment

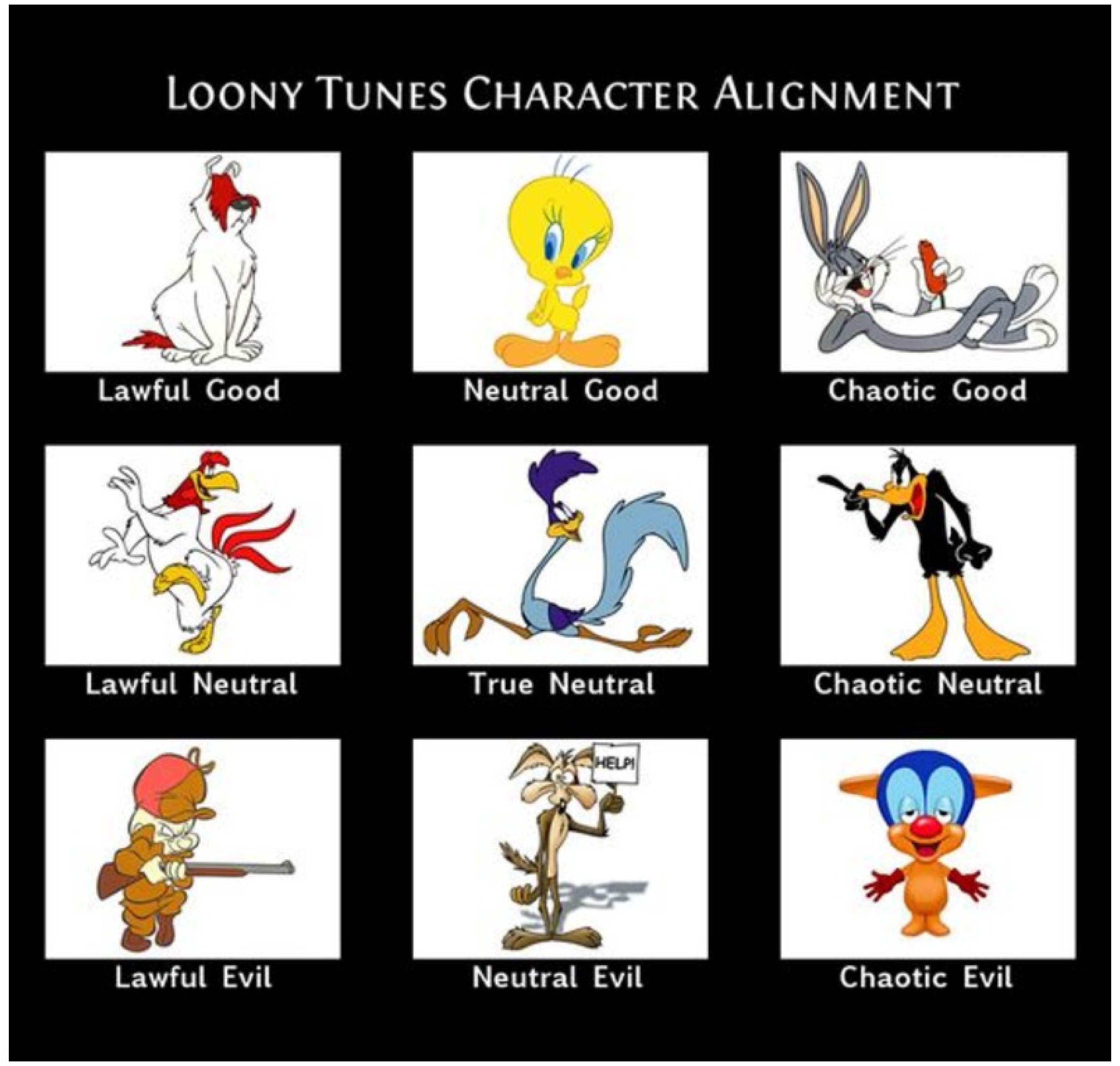
```
0 0 0 0 0 0 1 2 3 4 5 6 7 7
v _ _ _ _ _ A T G T T A T _
w A T C G T A _ _ _ _ _ _ T
0 1 2 3 4 5 6 6 6 6 6 6 6 7
```

- Path: (0,0), (0,1), (0,2), (0,3), (0,4), (0,5), (1,6), (2,6), (3,6), (4,6), (5,6), (6,6), (7,6), (7,7)



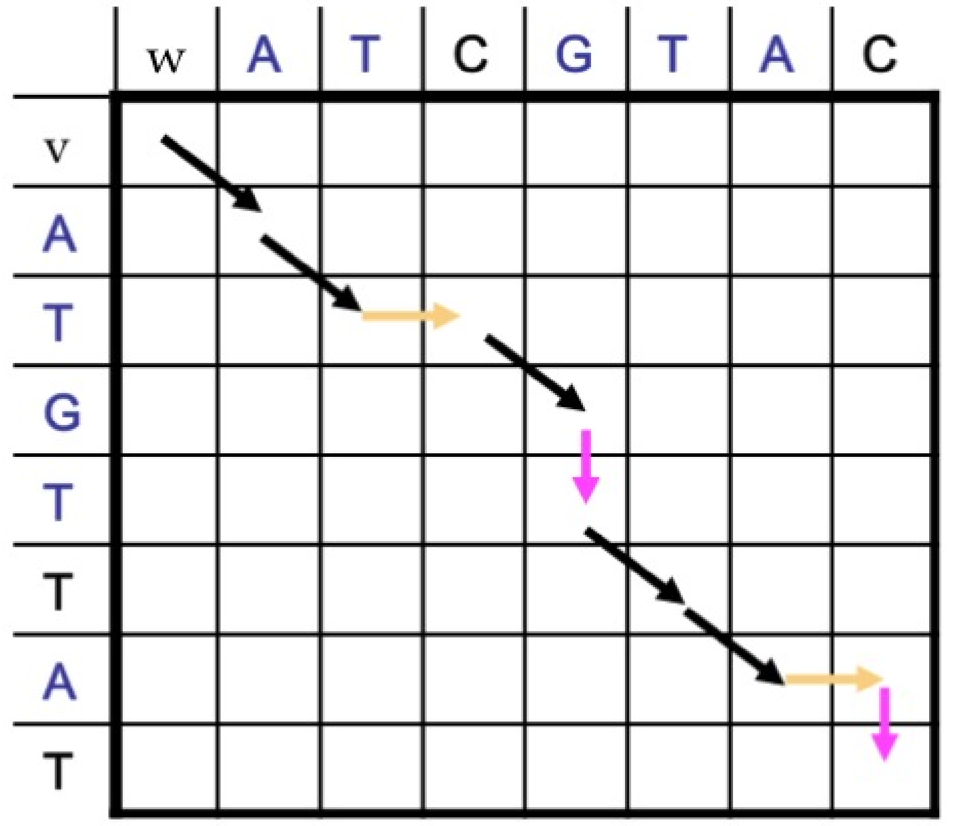
# What makes a good alignment?

- Using as many diagonal segments (matches) as possible
- The end of a good alignment from (j...k) begins with a good alignment from (i..j)
- Same as Manhattan Tourist problem, where ***sites*** are only on the diagonal streets!
- Set diagonal street weights = 1, and horizontal and vertical weights = 0



# Alignment: Dynamic Program

$$s_{i,j} = \max\begin{cases} s_{i-1,j-1} + 1 \quad \mathit{if}\ v_i = w_i & \searrow \\ \quad s_{i-1,j} & \downarrow \\ \quad s_{i,j-1} & \rightarrow \end{cases}$$



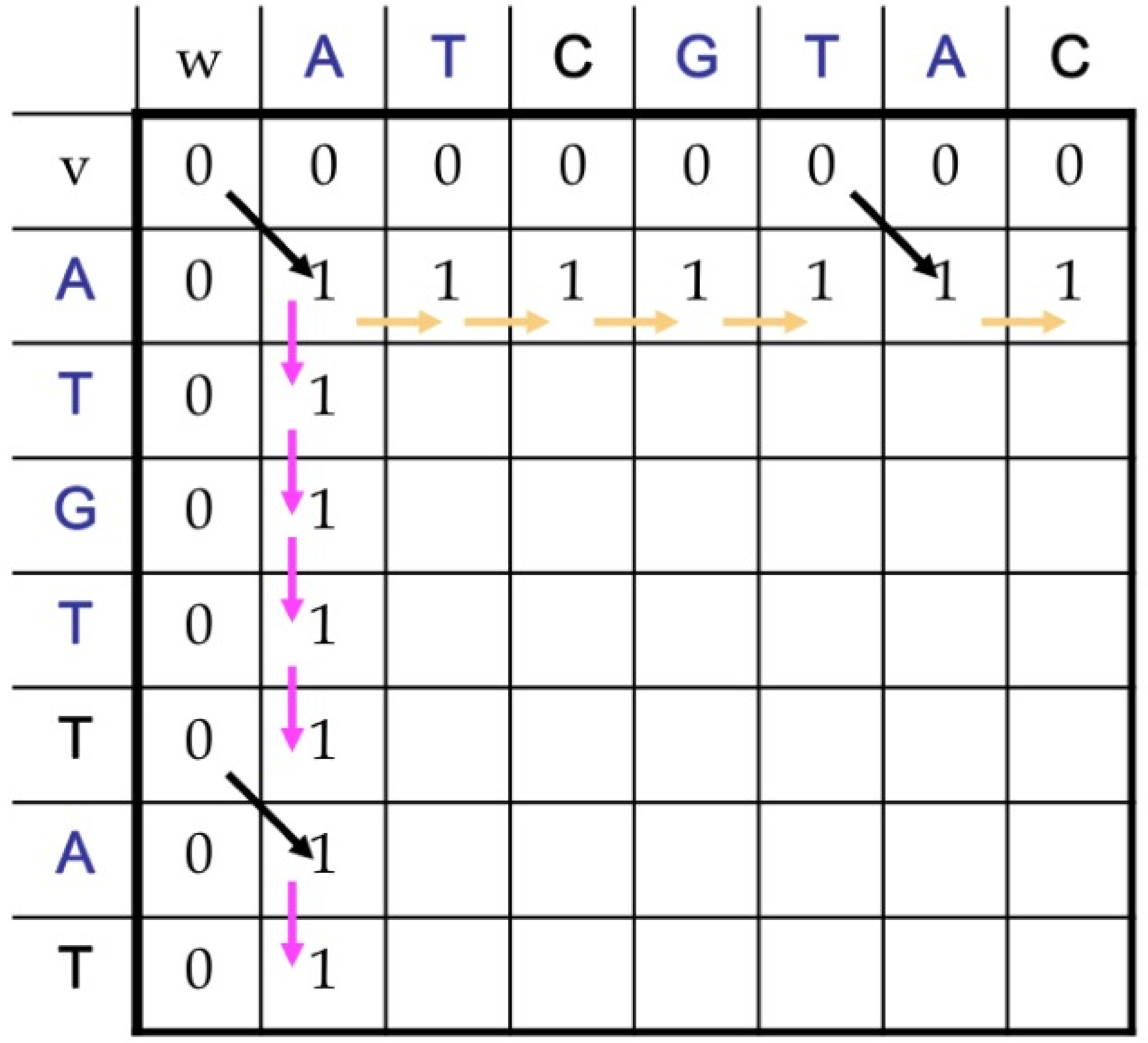
# Step 1

Initialize 1st row and 1st column to all zeroes.

| | w | A | T | C | G | T | A | C |
|---|---|---|---|---|---|---|---|---|
| v | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 0 | | | | | | | |
| T | 0 | | | | | | | |
| G | 0 | | | | | | | |
| T | 0 | | | | | | | |
| T | 0 | | | | | | | |
| A | 0 | | | | | | | |
| T | 0 | | | | | | | |

# Step 2

Evaluate recursion for next row and/or next column



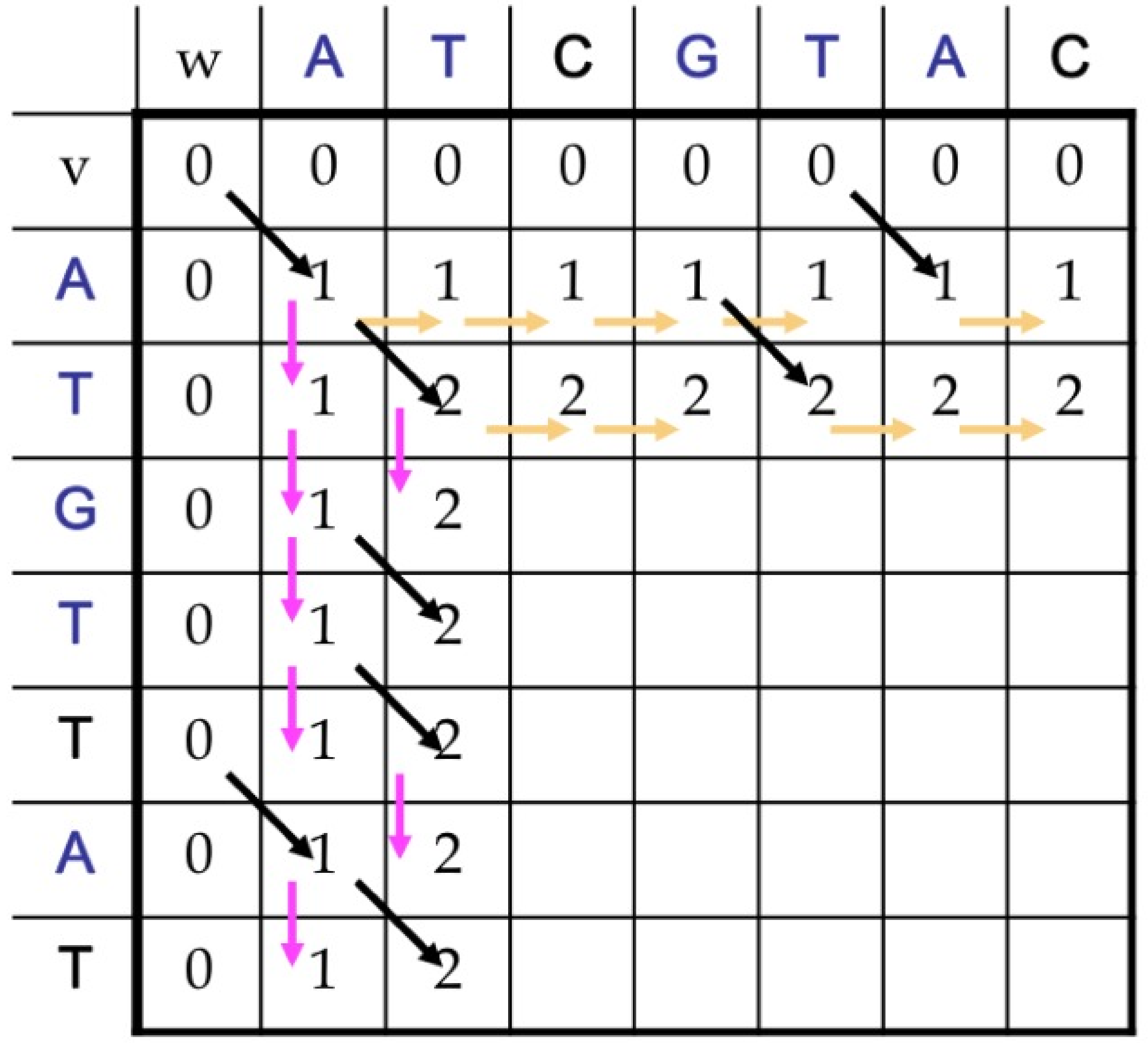
| | w | A | T | C | G | T | A | C |
|---|---|---|---|---|---|---|---|---|
| v | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| T | 0 | 1 | | | | | | |
| G | 0 | 1 | | | | | | |
| T | 0 | 1 | | | | | | |
| T | 0 | 1 | | | | | | |
| A | 0 | 1 | | | | | | |
| T | 0 | 1 | | | | | | |

$$s_{i,j} = \max\begin{cases} s_{i-1,j-1}+1 & if\ v_i = w_j \searrow \\ s_{i-1,j} & \downarrow \\ s_{i,j-1} & \rightarrow \end{cases}$$

First Previous Next Notebook

# Step 3

Continue recursion for next row and/or next column

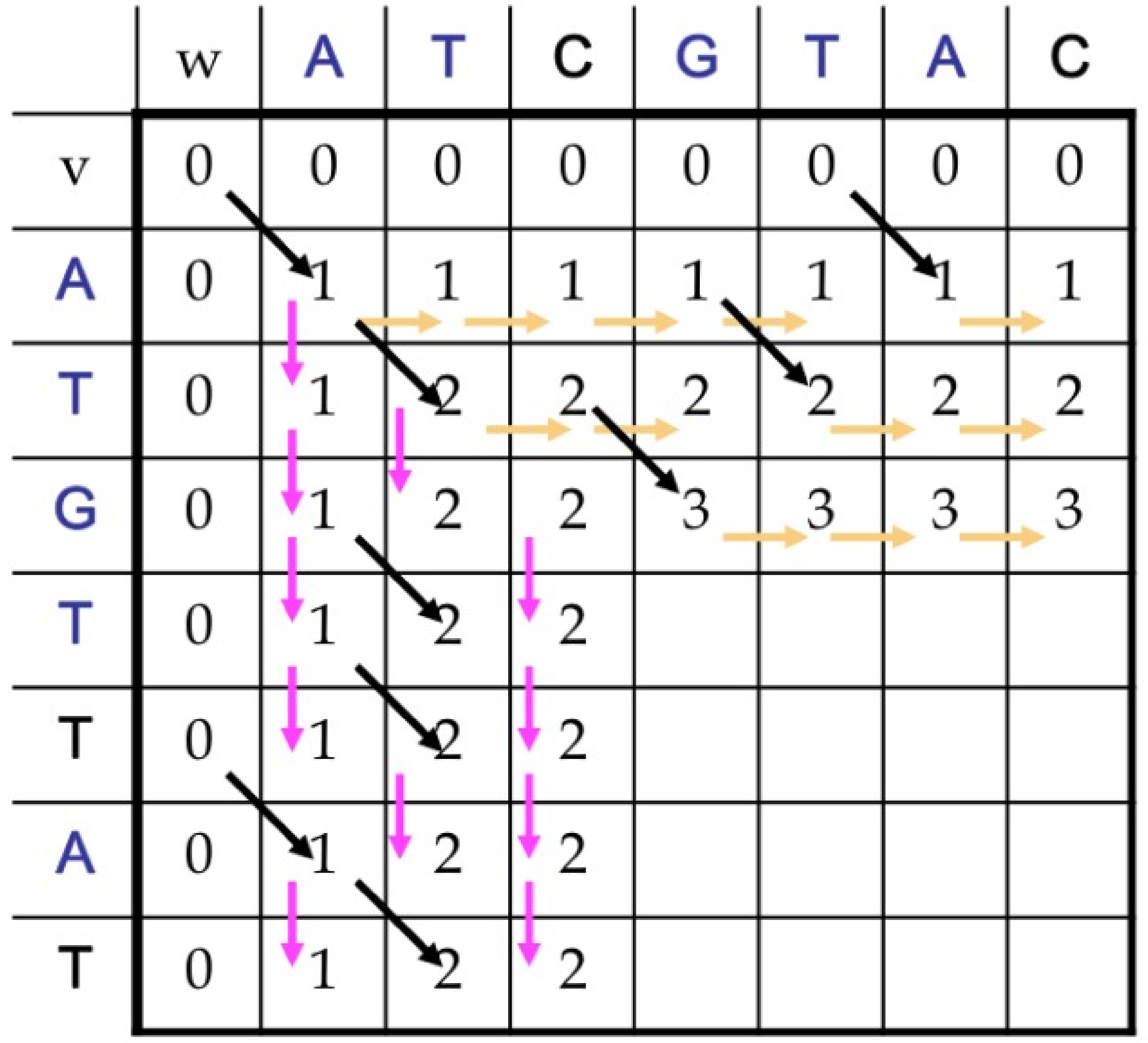


|   | w | A | T | C | G | T | A | C |
|---|---|---|---|---|---|---|---|---|
| v | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| T | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 2 |
| G | 0 | 1 | 2 |   |   |   |   |   |
| T | 0 | 1 | 2 |   |   |   |   |   |
| T | 0 | 1 | 2 |   |   |   |   |   |
| A | 0 | 1 | 2 |   |   |   |   |   |
| T | 0 | 1 | 2 |   |   |   |   |   |

$$s_{i,j} = \max\begin{cases} s_{i-1,j-1}+1 & \text{if } v_i = w_j \quad \searrow \\ s_{i-1,j} & \downarrow \\ s_{i,j-1} & \rightarrow \end{cases}$$

# Step 4

Then one more row and/or column

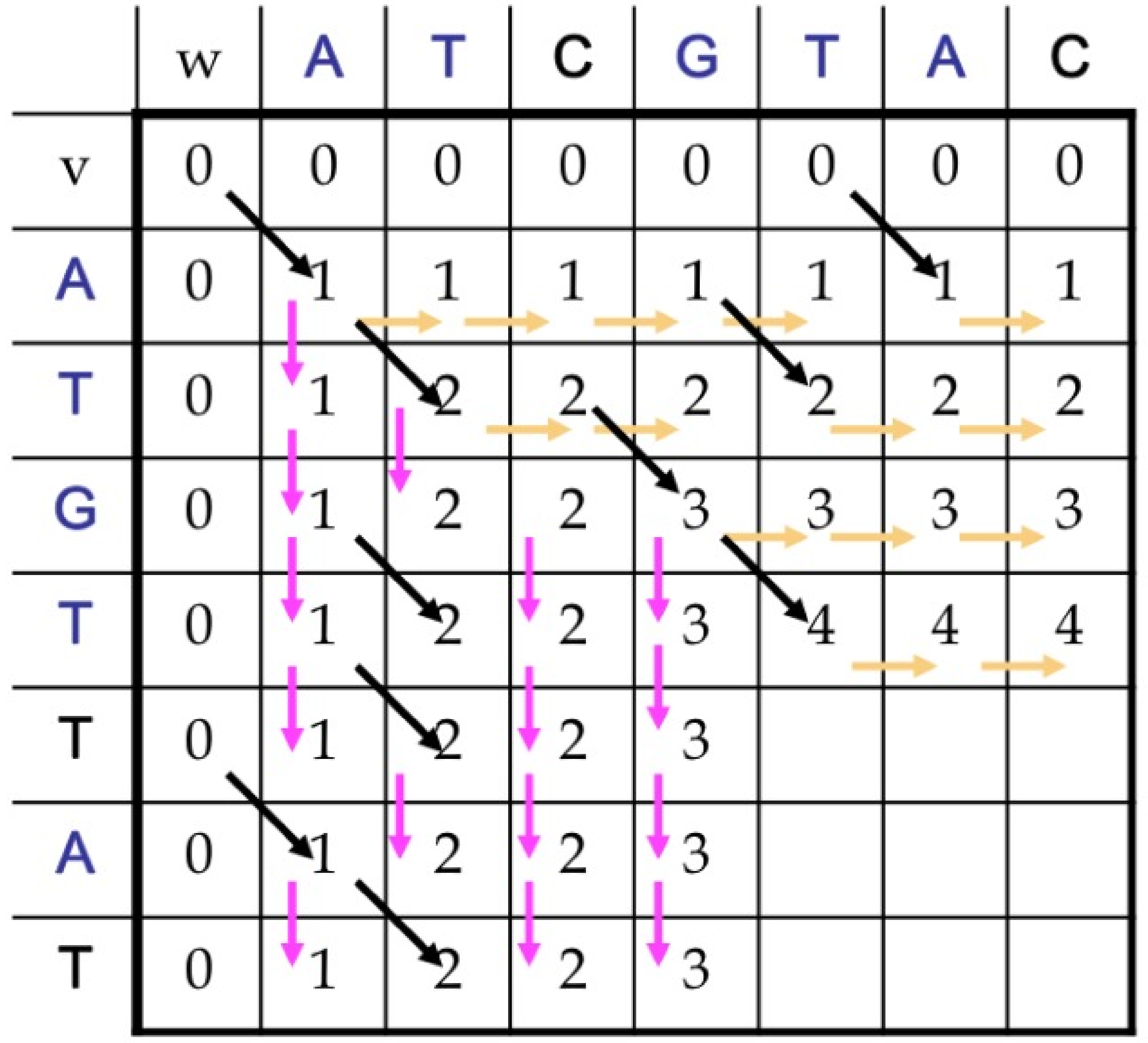


|   | w | A | T | C | G | T | A | C |
|---|---|---|---|---|---|---|---|---|
| v | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| T | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 2 |
| G | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3 |
| T | 0 | 1 | 2 | 2 |  |  |  |  |
| T | 0 | 1 | 2 | 2 |  |  |  |  |
| A |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

$$s_{i,j} = \max\begin{cases} s_{i-1,j-1}+1 & \mathit{if}\ v_i = w_j \ \searrow \\ s_{i-1,j} & \downarrow \\ s_{i,j-1} & \rightarrow \end{cases}$$

# Step 5

And so on…

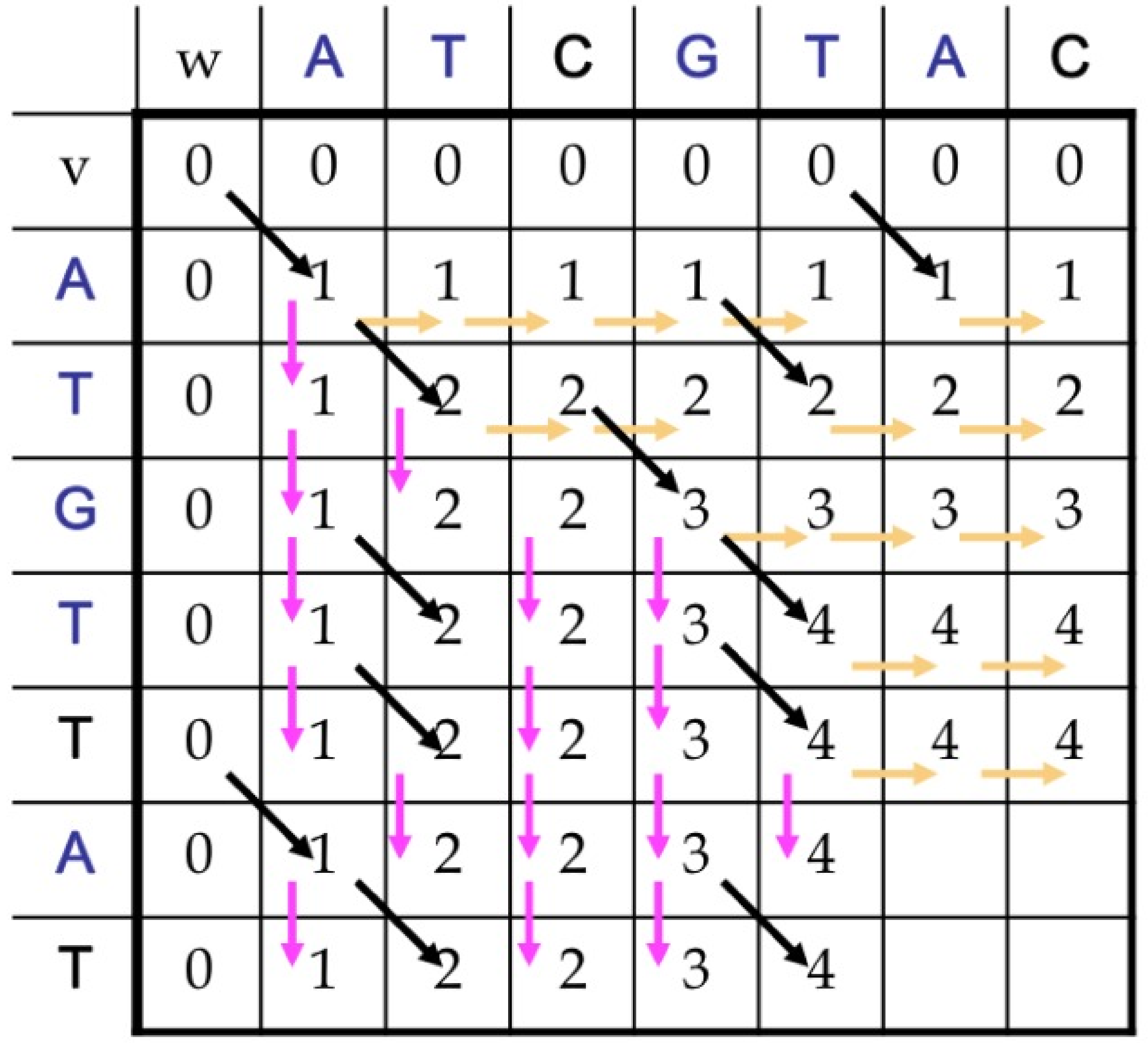


| | w | A | T | C | G | T | A | C |
|---|---|---|---|---|---|---|---|---|
| v | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| T | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 2 |
| G | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3 |
| T | 0 | 1 | 2 | 2 | 3 | 4 | 4 | 4 |
| T | 0 | 1 | 2 | 2 | 3 | | | |
| A | 0 | 1 | 2 | 2 | 3 | | | |
| T | 0 | 1 | 2 | 2 | 3 | | | |

$$s_{i,j} = \max \begin{cases} s_{i-1,j-1} + 1 & \textit{if } v_i = w_j \searrow \\ s_{i-1,j} & \downarrow \\ s_{i,j-1} & \rightarrow \end{cases}$$

# Step 6

And so on...

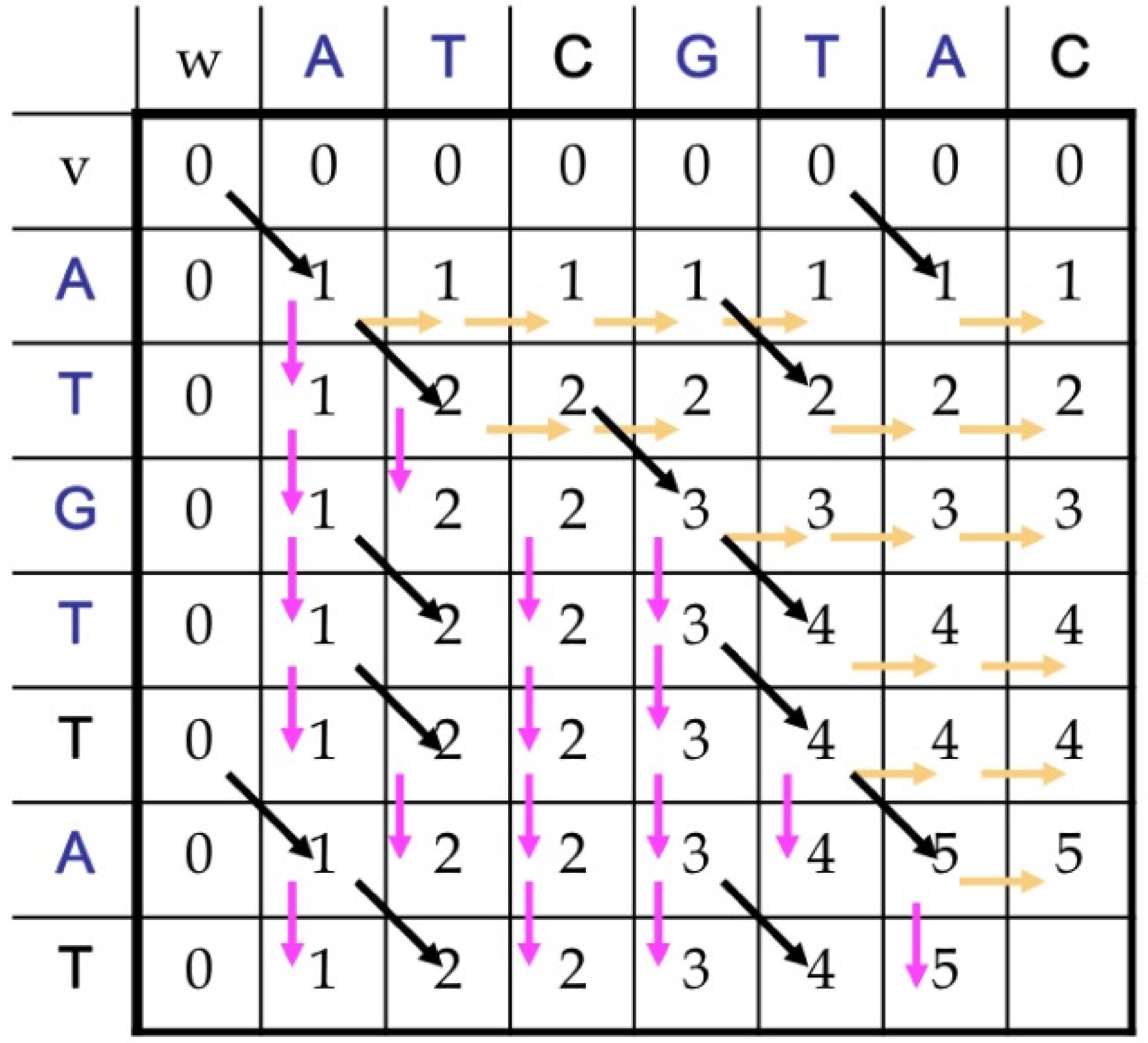


|   | w | A | T | C | G | T | A | C |
|---|---|---|---|---|---|---|---|---|
| v | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| T | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 2 |
| G | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3 |
| T | 0 | 1 | 2 | 2 | 3 | 4 | 4 | 4 |
| T | 0 | 1 | 2 | 2 | 3 | 4 | 4 | 4 |
| A | 0 | 1 | 2 | 2 | 3 | 4 |   |   |
| T | 0 | 1 | 2 | 2 | 3 | 4 |   |   |

$$s_{i,j} = \max \begin{cases} s_{i-1,j-1} + 1 & if\ v_i = w_j \ \searrow \\ s_{i-1,j} & \downarrow \\ s_{i,j-1} & \rightarrow \end{cases}$$

# Step 7

Getting closer

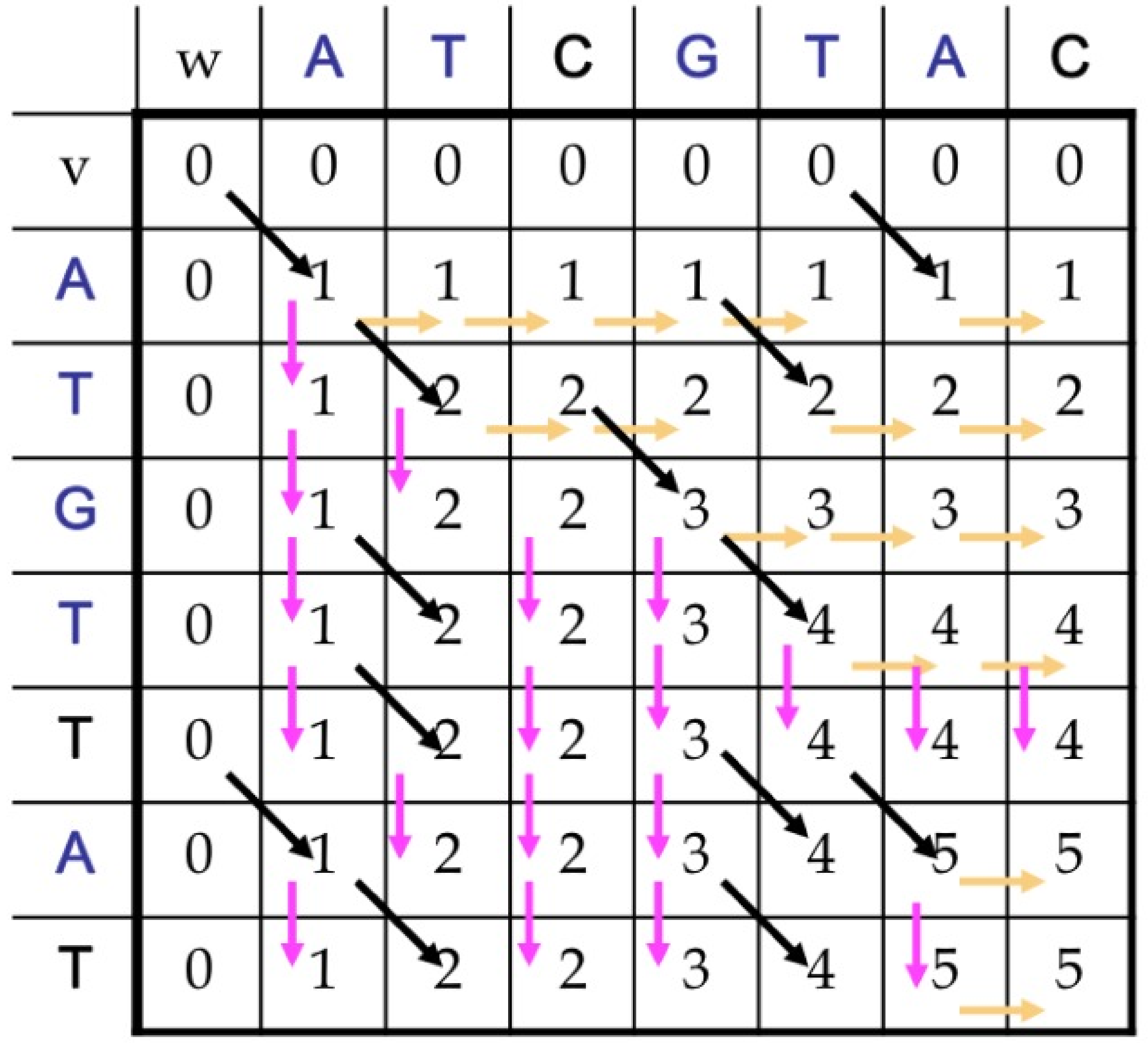


|   | w | A | T | C | G | T | A | C |
|---|---|---|---|---|---|---|---|---|
| v | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| T | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 2 |
| G | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3 |
| T | 0 | 1 | 2 | 2 | 3 | 4 | 4 | 4 |
| T | 0 | 1 | 2 | 2 | 3 | 4 | 4 | 4 |
| A | 0 | 1 | 2 | 2 | 3 | 4 | 5 | 5 |
| T | 0 | 1 | 2 | 2 | 3 | 4 | 5 |   |

$$s_{i,j} = \max \begin{cases} s_{i-1,j-1} + 1 & \text{if } v_i = w_j \quad \searrow \\ s_{i-1,j} & \downarrow \\ s_{i,j-1} & \rightarrow \end{cases}$$

# Step 8

Until we reach the last row and column

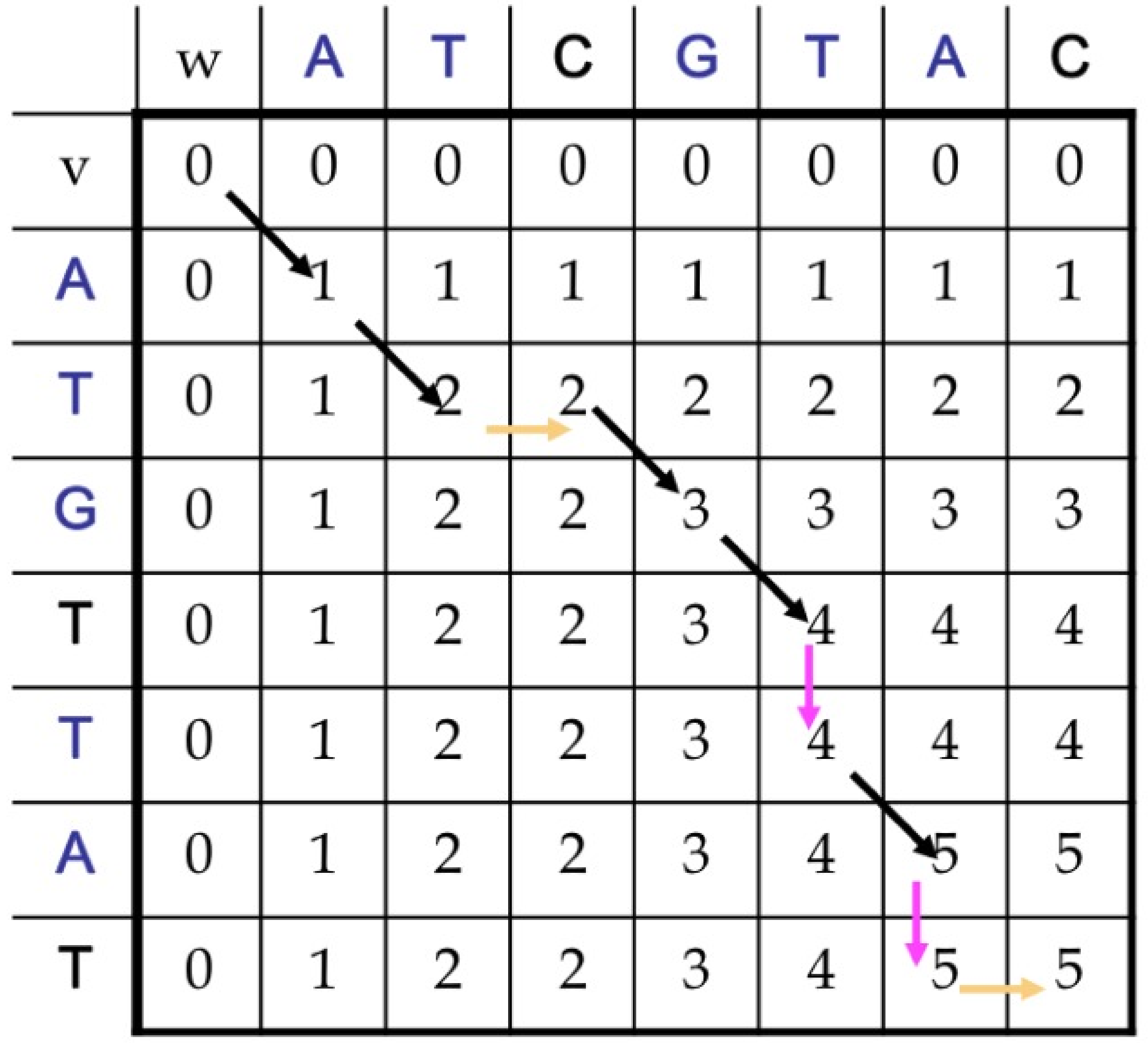


|   | w | A | T | C | G | T | A | C |
|---|---|---|---|---|---|---|---|---|
| v | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| T | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 2 |
| G | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3 |
| T | 0 | 1 | 2 | 2 | 3 | 4 | 4 | 4 |
| T | 0 | 1 | 2 | 2 | 3 | 4 | 4 | 4 |
| A | 0 | 1 | 2 | 2 | 3 | 4 | 5 | 5 |
| T | 0 | 1 | 2 | 2 | 3 | 4 | 5 | 5 |

$$s_{i,j} = \max \begin{cases} s_{i-1,j-1} + 1 \quad if\ v_i = w_j \searrow \\ s_{i-1,j} \quad \downarrow \\ s_{i,j-1} \quad \rightarrow \end{cases}$$

# Finally

We reach the end, which corresponds to an LCS of length 5



|   | w | A | T | C | G | T | A | C |
|---|---|---|---|---|---|---|---|---|
| v | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| T | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 2 |
| G | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3 |
| T | 0 | 1 | 2 | 2 | 3 | 4 | 4 | 4 |
| T | 0 | 1 | 2 | 2 | 3 | 4 | 4 | 4 |
| A | 0 | 1 | 2 | 2 | 3 | 4 | 5 | 5 |
| T | 0 | 1 | 2 | 2 | 3 | 4 | 5 | 5 |

$$s_{i,j} = \max \begin{cases} s_{i-1,j-1} + 1 & \text{if } v_i = w_j \searrow \\ s_{i-1,j} & \downarrow \\ s_{i,j-1} & \rightarrow \end{cases}$$

W = ATCGT-A-C
V = AT-GTTAT-

Our answer includes both an optimal score, and a path back to find the alignment

# LCS Code

Let's see how well the code matches the approach we sketched out...

```
from numpy import *

def findLCS(v, w):
    score = zeros((len(v)+1,len(w)+1), dtype="int32")
    backt = zeros((len(v)+1,len(w)+1), dtype="int32")
    for i in xrange(1,len(v)+1):
        for j in xrange(1,len(w)+1):
            # find best score at each vertex
            if (v[i-1] == w[j-1]):
                score[i,j], backt[i,j] = max((score[i-1,j-1]+1,3), (score[i-1,j],1), (score[i,j-1],2))
            else:
                score[i,j], backt[i,j] = max((score[i-1,j],1), (score[i,j-1],2))
    return score, backt

s, b = findLCS("ATGTTAT","ATCGTAC")
for i in xrange(len(s)):
    print "%10s %-20s    %12s %-20s" % ('' if i else 'score =', str(s[i]), '' if i else 'backtrack =', str(b[i]))
```

```
   score = [0 0 0 0 0 0 0 0]         backtrack = [0 0 0 0 0 0 0 0]
           [0 1 1 1 1 1 1 1]                     [0 3 2 2 2 2 3 2]
           [0 1 2 2 2 2 2 2]                     [0 1 3 2 2 3 2 2]
           [0 1 2 2 3 3 3 3]                     [0 1 1 2 3 2 2 2]
           [0 1 2 2 3 4 4 4]                     [0 1 3 2 1 3 2 2]
           [0 1 2 2 3 4 4 4]                     [0 1 3 2 1 3 2 2]
           [0 1 2 2 3 4 5 5]                     [0 3 1 2 1 1 3 2]
           [0 1 2 2 3 4 5 5]                     [0 1 3 2 1 3 1 2]
```

- The same score matrix that we found by hand
- *"backtrack"* keeps track of the arrows that we used

# Backtracking

Our score table kept track of the longest common subsequence so far. How do we figure out what the subsequence is?

The second "arrow" table kept track of the decisions we made... and we'll use it to backtrack to our answer.

In our example we used arrows {↓, →, ↘}, which were represented in our matrix as {1,2,3} respectively. This numbering is arbitrrary, except that it does break ties in our implementation (matches > *w* deletions > *w* insertions).

Next we need code that finds a path from the end of our strings to the beginning using our *arrow* matrix

# Code to extract our answer

We can write a simple recursive routine to return along the path of arrows that led to our best score.

```
def LCS(b,v,i,j):
    if ((i == 0) and (j == 0)):
        return ''
    if (b[i,j] == 3):
        result = LCS(b,v,i-1,j-1)
        result = result + v[i-1]
        return result
    else:
        if (b[i,j] == 2):
            return LCS(b,v,i,j-1)
        else:
            return LCS(b,v,i-1,j)

print LCS(b,"ATGTTAT",b.shape[0]-1,b.shape[1]-1)
```

```
ATGTA
```

- Notice that we only need one of *v* or *w* since both contain the LCS
- Perhaps we would like to get more than just the LCS; for example, the correpsonding alignment.

# An alignment of *v* and *w*

```
def Alignment(b,v,w,i,j):
    if ((i == 0) and (j == 0)):
        return ['','']
    if (b[i,j] == 3):
        result = Alignment(b,v,w,i-1,j-1)
        result[0] += v[i-1]
        result[1] += w[j-1]
        return result
    else:
        if (b[i,j] == 2):
            result = Alignment(b,v,w,i,j-1)
            result[0] += "_"
            result[1] += w[j-1]
            return result
        else:
            result = Alignment(b,v,w,i-1,j)
            result[0] += v[i-1]
            result[1] += "_"
            return result

align = Alignment(b,"ATGTTAT","ATCGTAC",b.shape[0]-1,b.shape[1]-1)
print "v =", align[0]
print "w =", align[1]
```

```
v = AT_GTTAT_
w = ATCG_TA_C
```

# Alignment with a Scoring Matrix

- Rather *edit distance* one could use a table with costs for every symbol aligned to any other
- Scoring matrices allow alignments to consider biological constraints
- Alignments can be thought of as two sequences that differ due to mutations.
- Some types of mutations are more common, or have little effect on the protein's function, therefore some mismatch penalties, $\delta(v_i, w_j)$, should be less harsh than others.

|   | A | R | N | K |
|---|---|---|---|---|
| A | 5 | -2 | -1 | -1 |
| R | - | 7 | -1 | 3 |
| N | - | - | 7 | 0 |
| K | - | - | - | 6 |

**Example:**

- Although R (arginine) and K (lysine) are different amino acids, they might still have a positive score.
- Why? They are both positively charged amino acids and hydrophillic implying such a substitution may not greatly change function of protein.

AKRANR
KAAANK
**-1 + (-1) + (-2) + 5 + 7 + 3 = 11**

# Functional Conservation

- Amino acid changes that tend to preserve the electro-chemical properties of the original residue
  - Polar to polar (aspartate → glutamate)
  - Nonpolar to nonpolar (alanine → valine)
  - Similarly behaving residues (leucine → isoleucine)
- Common Amino acid substitution matrices
  - PAM
  - BLOSUM
- DNA substitution matrices
  - DNA is less conserved than protein sequences
  - Less effective to compare coding regions at nucleotide level

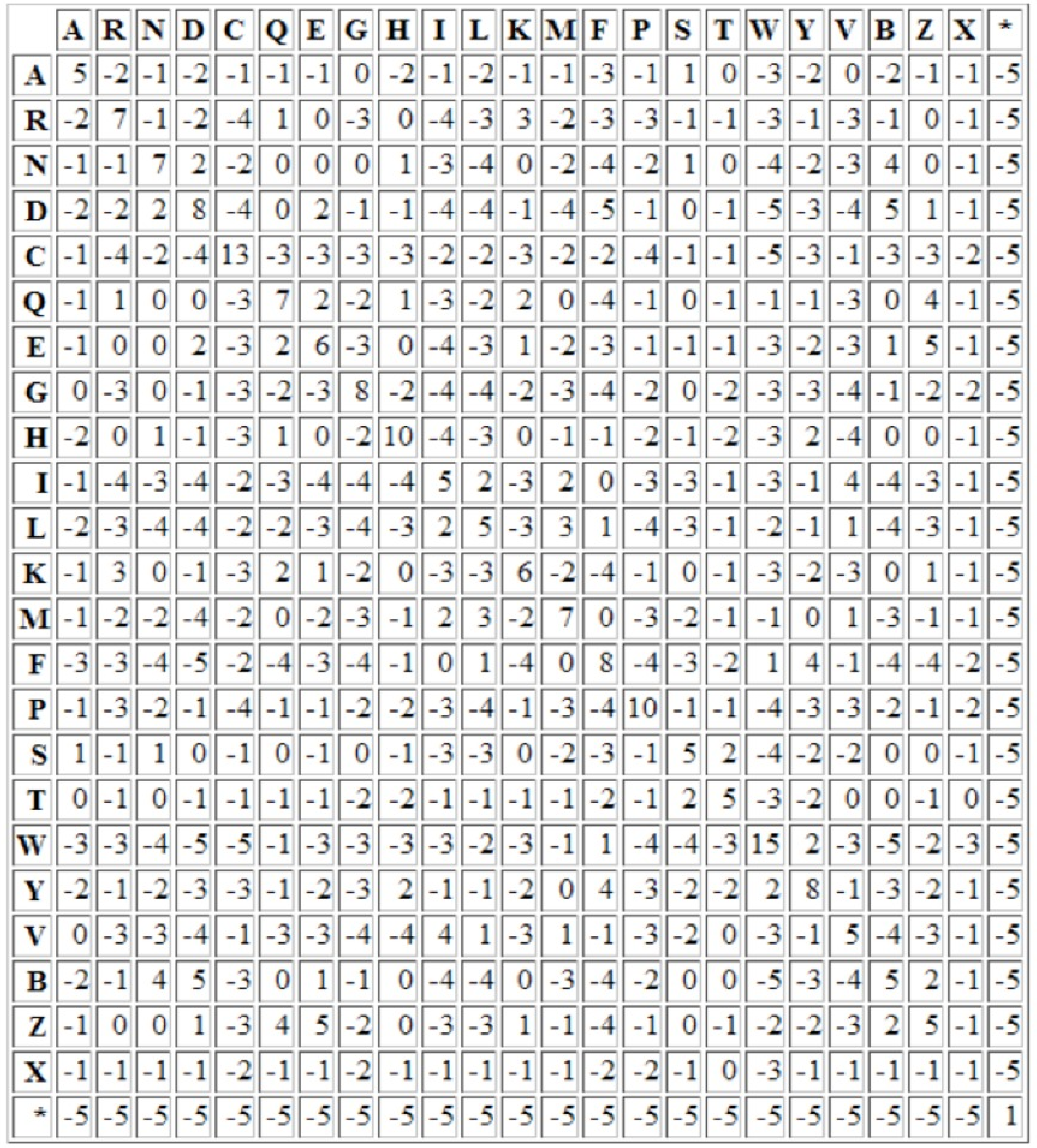
# PAM

- **P**oint **A**ccepted **M**utation (Dayhoff et al.)
- 1 PAM = $\text{PAM}_1$ = 1% average change of all amino acid positions
  - After 100 PAMs of evolution, not every residue will have changed
    - some residues may have mutated several times
    - some residues may have returned to their original state
    - some residues may not changed at all
- $\text{PAM}_X \sim (\text{PAM}_1)^X$
- $\text{PAM}_{250}$ is a widely used scoring matrix

| | Ala | Arg | Asn | Asp | Cys | Gln | Glu | Gly | His | Ile | Leu | Lys | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | R | N | D | C | Q | E | G | H | I | L | K | ... |
| Ala A | 13 | 6 | 9 | 9 | 5 | 8 | 9 | 12 | 6 | 8 | 6 | 7 | ... |
| Arg R | 3 | 17 | 4 | 3 | 2 | 5 | 3 | 2 | 6 | 3 | 2 | 9 | |
| Asn N | 4 | 4 | 6 | 7 | 2 | 5 | 6 | 4 | 6 | 3 | 2 | 5 | |
| Asp D | 5 | 4 | 8 | 11 | 1 | 7 | 10 | 5 | 6 | 3 | 2 | 5 | |
| Cys C | 2 | 1 | 1 | 1 | 52 | 1 | 1 | 2 | 2 | 2 | 1 | 1 | |
| Gln Q | 3 | 5 | 5 | 6 | 1 | 10 | 7 | 3 | 7 | 2 | 3 | 5 | |
| ... | | | | | | | | | | | | | |
| Trp W | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | |
| Tyr Y | 1 | 1 | 2 | 1 | 3 | 1 | 1 | 1 | 3 | 2 | 2 | 1 | |
| Val V | 7 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 5 | 4 | 15 | 10 | |

# BLOSUM

- **Blo**ck **Su**bstitution **M**atrix
- Scores derived from observations of the frequencies of substitutions in blocks of local alignments in related proteins
- Matrix name indicates evolutionary distance
- BLOSUM50 was created using actual sequences sharing no more than 50% identity



| | A | R | N | D | C | Q | E | G | H | I | L | K | M | F | P | S | T | W | Y | V | B | Z | X | * |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 5 | -2 | -1 | -2 | -1 | -1 | -1 | 0 | -2 | -1 | -2 | -1 | -1 | -3 | -1 | 1 | 0 | -3 | -2 | 0 | -2 | -1 | -1 | -5 |
| R | -2 | 7 | -1 | -2 | -4 | 1 | 0 | -3 | 0 | -4 | -3 | 3 | -2 | -3 | -3 | -1 | -1 | -3 | -1 | -3 | -1 | 0 | -1 | -5 |
| N | -1 | -1 | 7 | 2 | -2 | 0 | 0 | 0 | 1 | -3 | -4 | 0 | -2 | -4 | -2 | 1 | 0 | -4 | -2 | -3 | 4 | 0 | -1 | -5 |
| D | -2 | -2 | 2 | 8 | -4 | 0 | 2 | -1 | -1 | -4 | -4 | -1 | -4 | -5 | -1 | 0 | -1 | -5 | -3 | -4 | 5 | 1 | -1 | -5 |
| C | -1 | -4 | -2 | -4 | 13 | -3 | -3 | -3 | -3 | -2 | -2 | -3 | -2 | -2 | -4 | -1 | -1 | -5 | -3 | -1 | -3 | -3 | -2 | -5 |
| Q | -1 | 1 | 0 | 0 | -3 | 7 | 2 | -2 | 1 | -3 | -2 | 2 | 0 | -4 | -1 | 0 | -1 | -1 | -1 | -3 | 0 | 4 | -1 | -5 |
| E | -1 | 0 | 0 | 2 | -3 | 2 | 6 | -3 | 0 | -4 | -3 | 1 | -2 | -3 | -1 | -1 | -1 | -3 | -2 | -3 | 1 | 5 | -1 | -5 |
| G | 0 | -3 | 0 | -1 | -3 | -2 | -3 | 8 | -2 | -4 | -4 | -2 | -3 | -4 | -2 | 0 | -2 | -3 | -3 | -4 | -1 | -2 | -2 | -5 |
| H | -2 | 0 | 1 | -1 | -3 | 1 | 0 | -2 | 10 | -4 | -3 | 0 | -1 | -1 | -2 | -1 | -2 | -3 | 2 | -4 | 0 | 0 | -1 | -5 |
| I | -1 | -4 | -3 | -4 | -2 | -3 | -4 | -4 | -4 | 5 | 2 | -3 | 2 | 0 | -3 | -3 | -1 | -3 | -1 | 4 | -4 | -3 | -1 | -5 |
| L | -2 | -3 | -4 | -4 | -2 | -2 | -3 | -4 | -3 | 2 | 5 | -3 | 3 | 1 | -4 | -3 | -1 | -2 | -1 | 1 | -4 | -3 | -1 | -5 |
| K | -1 | 3 | 0 | -1 | -3 | 2 | 1 | -2 | 0 | -3 | -3 | 6 | -2 | -4 | -1 | 0 | -1 | -3 | -2 | -3 | 0 | 1 | -1 | -5 |
| M | -1 | -2 | -2 | -4 | -2 | 0 | -2 | -3 | -1 | 2 | 3 | -2 | 7 | 0 | -3 | -2 | -1 | -1 | 0 | 1 | -3 | -1 | -1 | -5 |
| F | -3 | -3 | -4 | -5 | -2 | -4 | -3 | -4 | -1 | 0 | 1 | -4 | 0 | 8 | -4 | -3 | -2 | 1 | 4 | -1 | -4 | -4 | -2 | -5 |
| P | -1 | -3 | -2 | -1 | -4 | -1 | -1 | -2 | -2 | -3 | -4 | -1 | -3 | -4 | 10 | -1 | -1 | -4 | -3 | -3 | -2 | -1 | -2 | -5 |
| S | 1 | -1 | 1 | 0 | -1 | 0 | -1 | 0 | -1 | -3 | -3 | 0 | -2 | -3 | -1 | 5 | 2 | -4 | -2 | -2 | 0 | 0 | -1 | -5 |
| T | 0 | -1 | 0 | -1 | -1 | -1 | -1 | -2 | -2 | -1 | -1 | -1 | -1 | -2 | -1 | 2 | 5 | -3 | -2 | 0 | 0 | -1 | 0 | -5 |
| W | -3 | -3 | -4 | -5 | -5 | -1 | -3 | -3 | -3 | -3 | -2 | -3 | -1 | 1 | -4 | -4 | -3 | 15 | 2 | -3 | -5 | -2 | -3 | -5 |
| Y | -2 | -1 | -2 | -3 | -3 | -1 | -2 | -3 | 2 | -1 | -1 | -2 | 0 | 4 | -3 | -2 | -2 | 2 | 8 | -1 | -3 | -2 | -1 | -5 |
| V | 0 | -3 | -3 | -4 | -1 | -3 | -3 | -4 | -4 | 4 | 1 | -3 | 1 | -1 | -3 | -2 | 0 | -3 | -1 | 5 | -4 | -3 | -1 | -5 |
| B | -2 | -1 | 4 | 5 | -3 | 0 | 1 | -1 | 0 | -4 | -4 | 0 | -3 | -4 | -2 | 0 | 0 | -5 | -3 | -4 | 5 | 2 | -1 | -5 |
| Z | -1 | 0 | 0 | 1 | -3 | 4 | 5 | -2 | 0 | -3 | -3 | 1 | -1 | -4 | -1 | 0 | -1 | -2 | -2 | -3 | 2 | 5 | -1 | -5 |
| X | -1 | -1 | -1 | -1 | -2 | -1 | -1 | -2 | -1 | -1 | -1 | -1 | -1 | -2 | -2 | -1 | 0 | -3 | -1 | -1 | -1 | -1 | -1 | -5 |
| * | -5 | -5 | -5 | -5 | -5 | -5 | -5 | -5 | -5 | -5 | -5 | -5 | -5 | -5 | -5 | -5 | -5 | -5 | -5 | -5 | -5 | -5 | -5 | 1 |

# Local vs. Global Alignment

- The *Global Alignment Problem* tries to find the highest scoring path between vertices (0,0) and (n,m) in the edit graph.
- The *Local Alignment Problem* tries to find the highest scoring subpath between all vertex pairs $(i_1, j_1)$ and $(i_2, j_2)$ in the edit graph where $i_2 > i_1$ and $j_2 > j_1$.
- In an edit graph with negatively-weighted scores, a Local Alignment may score higher than a Global Alignment

Example:

- Global Alignment

```
--T—-CC-C-AGT—-TATGT-CAGGGGACACG—A-GCATGCAGA-GAC
  |  || |  ||  | | | |||    || | | |  | ||||   |
AATTGCCGCC-GTCGT-T-TTCAG----CA-GTTATG—T-CAGAT--C
```

- Local Alignment finds longer conserved segment

```
                  tccCAGTTATGTCAGgggacacgagcatgcagagac
                     ||||||||||||
aattgccgccgtcgttttcagCAGTTATGTCAGatc
```

# Local Alignments: Why?

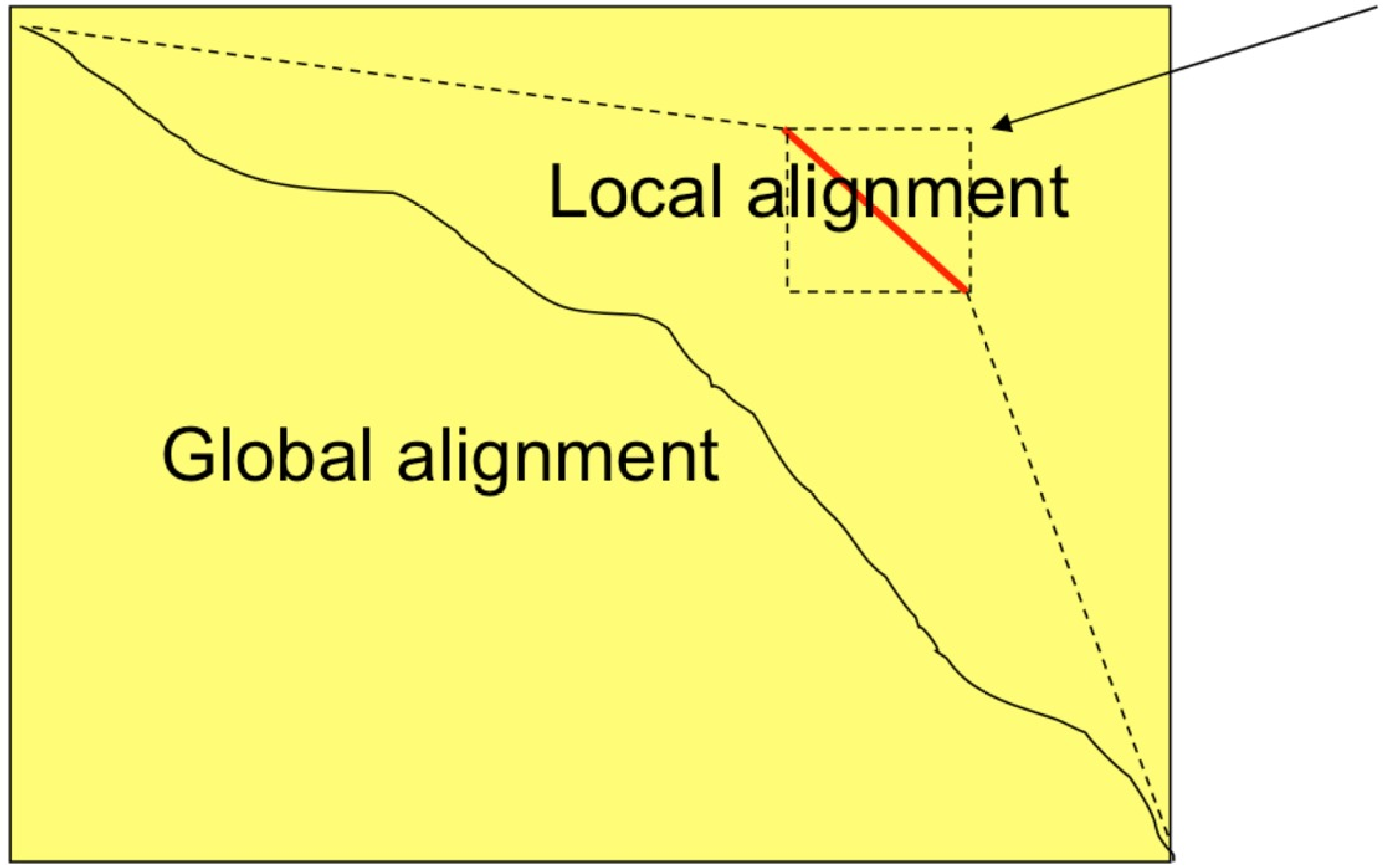
- Two genes in different species may be similar over short conserved regions and dissimilar over remaining regions.
- Example:
  - Homeobox genes have a short region called the homeodomain that is highly conserved between species.
  - A global alignment would not find the homeodomain because it would try to align the ENTIRE sequence

**Local Alignment Problem:**

- **Goal**: Find the best local alignment between two strings
- **Input**: Strings $v$, $w$ and scoring matrix $\delta$
- **Output**: Alignment of substrings of $v$ and $w$ whose alignment score is maximum among all possible alignment of all possible substrings

# Local Alignment Approach

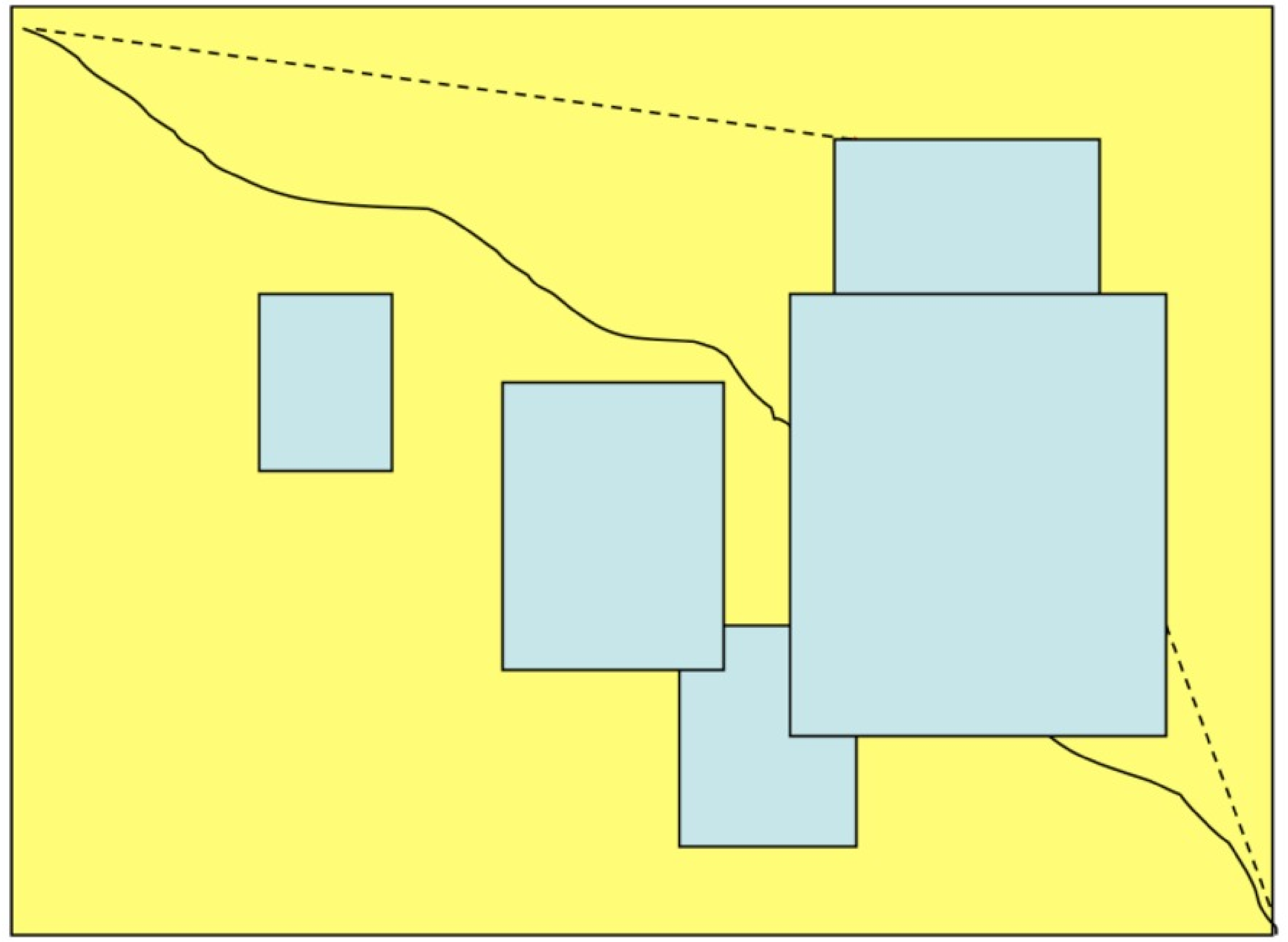
A local alignment is a subpath in a global alignment



Compute a “mini” Global Alignment to get Local

# Brute Force Local Alignment

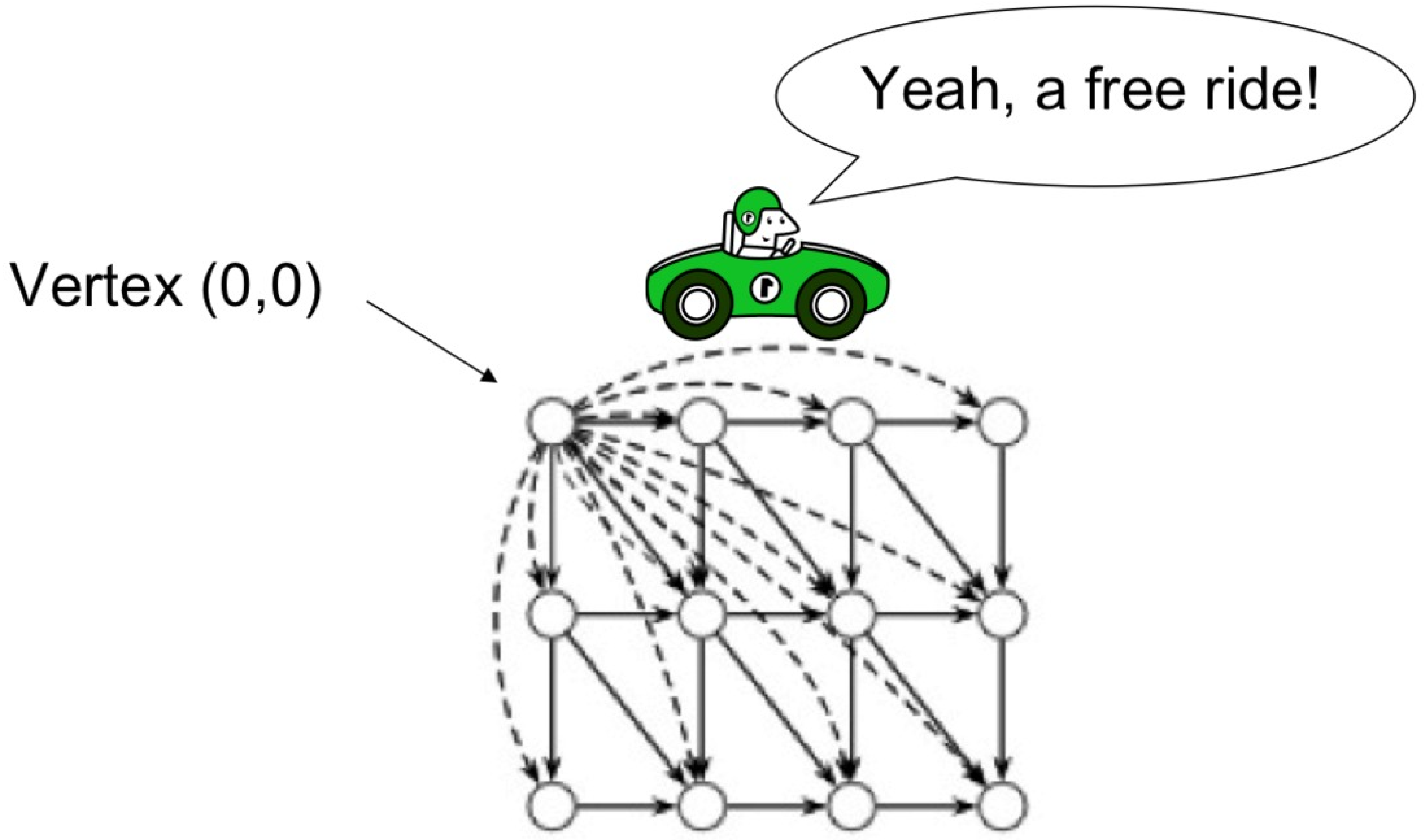
Find the best global alignment amoung all blocks $(i_1, j_1, i_2, j_2)$



- Long run time $O(n^4)$:
    - In the grid of size n x n there are $O(n^2)$ vertices $(i_1, j_1)$ that may serve as a source.
    - For each such vertex computing alignments from $(i_1, j_1)$ to $(i_2, j_2)$ takes $O(n^2)$ time.
- This can be remedied by giving free rides

# Local Alignment: Free Rides

- ***Key Ideas:*** Add extra edges to our graph, consider all scores in matrix



- The dashed edges represent a *free ride* from (0,0) to any other node
- The largest value of $s_{i,j}$ over the *whole score matrix* is the end point of the best local alignment (instead of $s_{n,m}$).

# The Local Alignment Recurrence

$$s_{i,j} = max \begin{cases} 0 \\ s_{i-1,j-1} + \delta\,(v_i,\, w_j) \\ s_{\,i-1,j} + \delta\,(v_i,\, -) \\ s_{\,i,j-1} + \delta\,(-,\, w_j) \end{cases}$$

Notice there is only this small change from the original recurrence of a Global Alignment

- The *zero* is our *free ride* that allows the node to restart with a score of 0 at any point
    - What does this imply?
- After solving for the entire score matrix, we then search for $s_{i,j}$ with the highest score, this is $(i_2, j_2)$
- We follow our back tracking matrix until we reach a *score* of 0, whose coordinate becomes $(i_1, j_1)$

# Next Time

- Alignment with Gap Penalities
- Multiple Alignment problem
- Can we do better than $O(MN)$?