Comparing biological sequences: sequence alignment

Motivation

The biological motivation for studying this problem comes from the fact that high degree of similarity of bimolecular sequences usually implies **significant structural and functional similarity**. Generally, such comparisons involves aligning sectionsof the two sequences in a way that exposes similarities between them.

Motivation

- The sequence similarity is also relevant in the context of understanding the **molecular basis of evolution**.
- It is well known that the closely related organisms have high similarity between their genomes.
- Study of conserved sequences among the organisms reveal past speciation and the structure of ancestral family trees and the role of mutation in the evolution of these trees.
- Studying similarities within the individual organisms in a species might also reveal whether certain individuals are prone to **inherited diseases**.
- There are many other examples from biology that illustrates the use of sequence similarity.

Alignment Example

- Consider the alignment between two protein sequences *v* and *w* (monkey sematotropin and its precursor in rainbow trout) .

- *v*: L E P V Q F L R S V F A N S L - V Y G T S Y S
- *w*: E Y P S Q T L - - I I S N S L M V - R N A - N

- The aligned pairs of symbols are written one above the other. Symbols in one sequence that is not aligned with symbols in the other are written opposing the symbol '-'. For example, 'RS' in sequence *v* constitutes a gap of two symbols. To maximize similarity of two sequences, we want to find alignments that have few gaps and align symbols that are identical or whose functions are highly similar.

Alignment Example

The dual perspective is to consider sequence *v* evolve into sequence *w* and we seek alignments with *minimum distance* or intuitively minimum number of evolutionary operations that converts *v* to *w*. In this perspective, an aligned symbol of *w* is *substituted* for its corresponding symbol in *v*, an unaligned symbol (that is, aligned with '-' symbol) of *v* is *deleted* and an unaligned symbol of *w* is *inserted*.

In the example, symbol 'E' of *w* is substituted for 'L' of *v*, and the symbols 'RS' in *v* constitute a *deletion gap*. Similarly, the symbol 'M' in *w* constitutes an *insertion gap* of one symbol. The symbol '-' is also referred to as an 'indel' standing for either insertion or deletion.

DNA Sequence Comparison: First Success Story

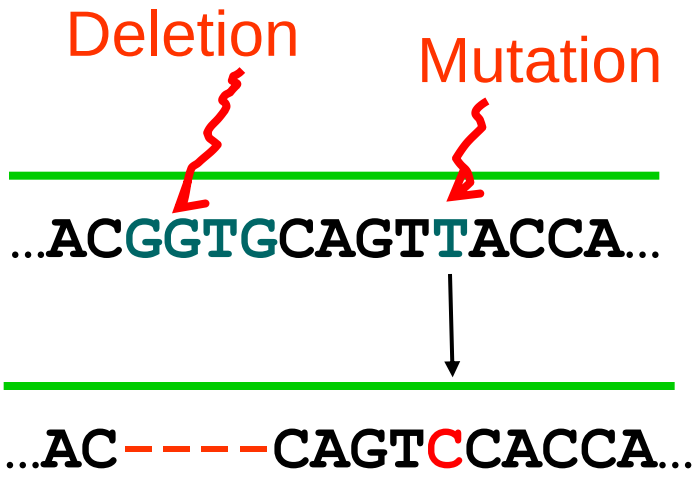
- Finding sequence similarities with genes of known function is a common approach to infer a newly sequenced gene's function

- In 1984 Russell Doolittle and colleagues found similarities between cancer-causing oncogene (**v-**sys in Simian Sarcoma Virus) and normal growth factor (PDGF) gene

Other Examples

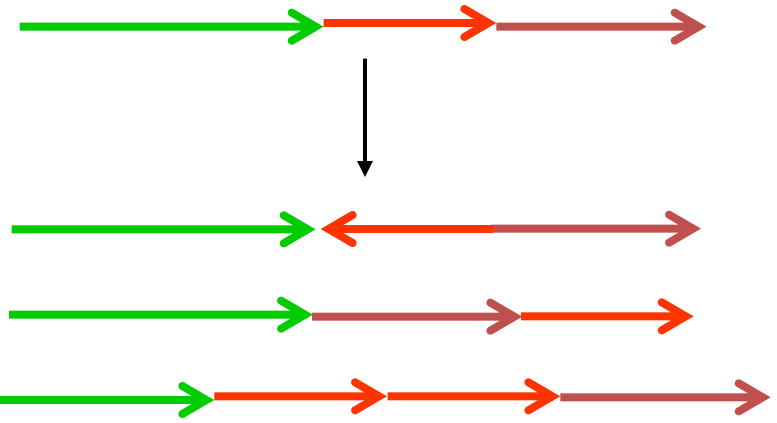
- The first successful sequencing of the genome of a living organism in 1995 of the bacterium *Haemophilus influenza rd* (Fleishmann et al, 1995).
- After that, researchers identified 1743 sites as prospective gene sites.
- In order to determine whether these sites are actually involved in protein synthesis, the coding regions were translated into amino acid sequences using the genetic code.
- The resulting amino acid sequences were then compared with a protein database that contains for each known protein the corresponding amino acid sequences.
- The search identified about 1007 close matches. Since the protein database annotated with functions, the close matches allowed coming up with strong conjectures about the functions of these genes.
- Identifying the similarity between ATP binding ion channels and the Cystic Fibrosis gene led to a modern hypothesis of CF.

Evolution at the DNA level

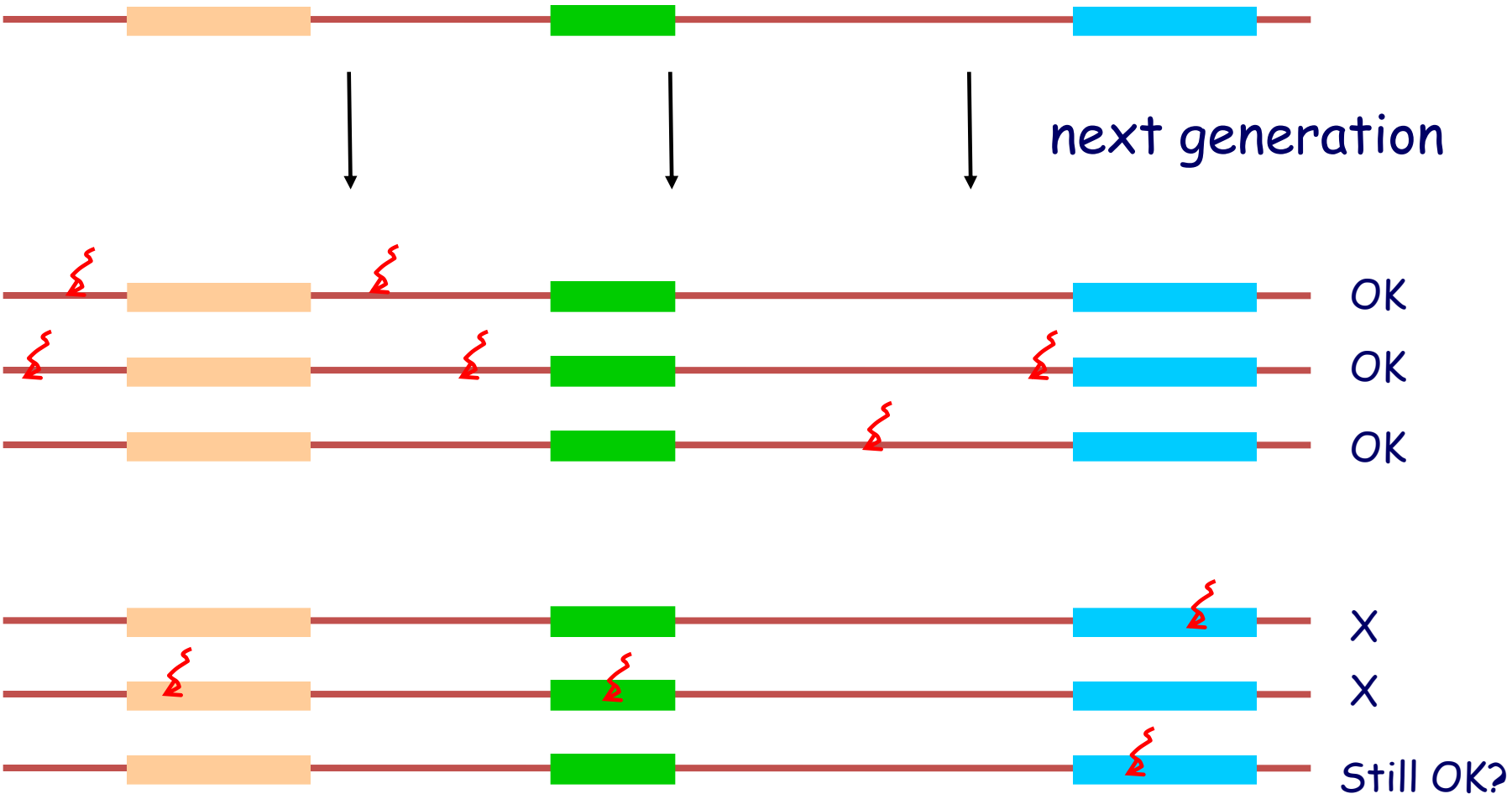


SEQUENCE EDITS

REARRANGEMENTS



Evolutionary Rates



Sequence Similarity

In these notes, we will talk about *sequence* ***similarity*** *or* ***alignment*** *of two sequences*. The sequences we are concerned with are DNA over the alphabet $\sum=(A,C,T,G)$, RNA over the alphabet $\sum=(A,C,U,G)$ or amino acid sequences (20 symbols) making up a protein molecule.

Goal of Sequence Alignment

- The goal of sequence alignment is to discover the possible evolution of sequences without actual knowledge of the evolutionary events.
- Naturally, the alignment with minimum number of operations involving minimum energy may be the Nature's choice. This transformation, as we will see, soon corresponds to ***edit distance*** between the sequences.

Alignment of Two Sequences

We will discuss the similarity algorithms with respect to . DNA , RNA or protein sequences. The three basic operations that take place in nature:

- A **deletion** operation, *D*.
- A **replacement or substitution** operation, *R*.
- An **insertion** operation, *I*.

Edit Operations

The two operations insertion and deletion are sometimes referred to as an indel operation when the direction of transformation is not known.

If the two characters are identical, it is called a match.

A simplified model of change in DNA sequence during evolution is to assume that substitution or indel events might have happened at any location in the sequence. Such events are also referred to as mutation of a DNA sequence.

(A *replacement* or *substitution* operation can be conceived of a *delete* operation followed by an *insert* operation. Thus the edit distance can be expressed only in terms of insert and delete operations. But. In the context of biological operations, substitution must be recognized as a distinct operation.)

The sequence of edit operations to transform a sequence *v* to another sequence *w* is called the edit transcript

Edit Distance

Levenshtein (1966) introduced edit distance between two strings as the minimum number of elementary operations (insertions, deletions, and substitutions) to transform one string into the other

$d(\boldsymbol{v},\boldsymbol{w})$ = MIN number of elementary operations to transform $\boldsymbol{v} \rightarrow \boldsymbol{w}$

Edit Distance vs Hamming Distance

Hamming distance always compares ***i***-th letter of ***v*** with ***i***-th letter of ***w***

v = ATATATAT
|| || || || || ||
w = TATATATA

Hamming distance:

$d(v, w)=8$

Computing Hamming distance is a trivial task.

Edit Distance vs Hamming Distance

Hamming distance always compares ***i***-th letter of **v** with ***i***-th letter of **w**

v = ATATATAT

w = TATATATA

Just one shift

Make it all line up

Edit distance may compare ***i***-th letter of **v** with ***j***-th letter of **w**

v = - ATATATAT

w = TATATATA-

Hamming distance:

$d(\mathbf{v}, \mathbf{w})=8$

Computing Hamming distance is a trivial task

Edit distance:

$d(\mathbf{v}, \mathbf{w})=2$

Computing edit distance is a non-trivial task

Edit Distance vs Hamming Distance

Hamming distance always compares i-th letter of **v** with i-th letter of **w**

```
v = ATATATAT
    |||  ||  |||
w = TATATATA
```

Edit distance may compare i-th letter of **v** with j-th letter of **w**

```
v = - ATATATAT
      |||  ||  ||
w = TATATATA
```

Hamming distance:

$d(\mathbf{v}, \mathbf{w})=8$

Edit distance:

$d(\mathbf{v}, \mathbf{w})=2$

(one insertion and one deletion)

How to find what j goes with what i ???

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 3rd G)
ATCCAT	→	(insert G before last A)
ATCCGAT		(Done)

Edit Distance: Example (cont'd)

TGCATAT → ATCCGAT in 4 steps

–TGCATAT → (insert A at front)

ATGCATAT → (delete 6th T)

ATGCATA– → (substitute G for 5th A)

ATGCGTA → (substitute C for 3rd G)

ATCCGAT (Done)

Can it be done in 3 steps???

Alignment

- Given a sequence *ATAGCCAT* and assume that a sequence of operations (*R*,*D*,*I*) have taken place as follows:

R	*D*	*I*
*ATAG**C**CAT*	*A**T**AGTCAT*	*AAGTC--AT*
*ATAG**T**CAT*	*A--AGTCAT*	*AAGTC**T**AT*

  - Biologists call these operations "**alignment**" and represent them by writing the two sequences, one over the other.

The Basic Operations

- If *a* and *b* are two distinct symbols, then the operations can be denoted by the ordered pair in any vertical column of the alignment as

  $R=(a,b)$, $D=(a,-)$ and $I=(-,b)$,

  where '– 'denote a null sequence or character.
- Obviously, (-,-) is a useless operation aligning null sequence with null sequence.
- Symbols pair that are identical in a vertical column represents matched symbols and sometime an operation *M (match)* is defined for this situation.

Alignment Example

- For this example, the combined effect of the three operations can be captured by the alignment

*A**T**AG**C**C-AT*
*A-AG**T**C**T**AT*

- In the evolutionary history, the accumulated changes may obscure the exact sequence of operations.
- E.g., the same final sequence may be obtained by the alignment that needs 5, rather than 3 operations:

*A**T**AG**C**C**AT**-*
*A-AG**T**C**TAT***

Formal Definition: Alignment

- Let *v* and *w* be two sequences of length *n* and *m* , respectively , over a finite alphabet ∑. An **alignment** maps the strings *v* and *w* into strings *v'* and *w'* respectively, that may contain indel ("-") characters such that

$$v' = w'$$

and removal of all indel characters leaves *v* and *w* intact. Let *l*= length of the alignment

Number of Alignments

- It is clear that $\max(n,m) \le l \le n+m$. The case *l*=*n*+*m* occurs when the alignment corresponds to deleting all characters in *v* followed by insertion of all characters of *w*.
- Let *f*(*i*,*j*) denote the number of alignments of one sequence of *i* letters with another of *j* letters. Then, it has been proved that

$$f(n,m) \approx (1+\sqrt{2})^{2n+1} n^{-1/2}$$

- For n=1000, ***f*(1000,1000)=** $10^{767.4}$ **alignments!** The number of elementary particles in the universe is about 10^{80}, and Avogadro's number is 10^{23}.

Edit Transcript

- A string over the operation alphabet (*R,I,D,M*) of length *l* that transforms *v* to *w* is called an ***edit transcript***.
- For the alignment:

A-TCCGAT-

TAT-C-ATC

The edit transcript is: *RIMDMDMMI* which converts *ATCCGAT* to *TATCATC.*

General definition of Edit Distance

The notion of a best alignment requires a more careful definition of some scoring or optimization criteria. We define a scoring function δ that assigns a cost to each possible assigned pair. Thus, if the sequences are over alphabet Σ, then

$\delta(a,b)$: cost for substituting or replacing '*a*' in *v* by '*b*' *in w*
$\delta(a,-)$: cost of deleting '*a*' *from v creating a gap in w.*
$\delta(-,b)$: cost of inserting '*b*' in *v* creating a *gap* in *v*

Thus, δ can be specified by a $|\Sigma|+1$ by $|\Sigma|+1$ table or matrix of real numbers. The score of an alignment between sequences *v* and *w* is the sum of the costs of the aligned pairs in it.

Technical comments on edit distance

- Symmetrical: $\delta(v,w) = \delta(w,v)$
  - Can "reverse the movie"
  - Substitution $a \rightarrow b$ becomes substitution $b \rightarrow a$
  - Insertion becomes deletion
  - Deletion becomes insertion
- "Parsimony" principle often used in computational biology
  - Simplest explanation for an observation
  - minimum number of edits = fewest mutations

Optimal Alignment

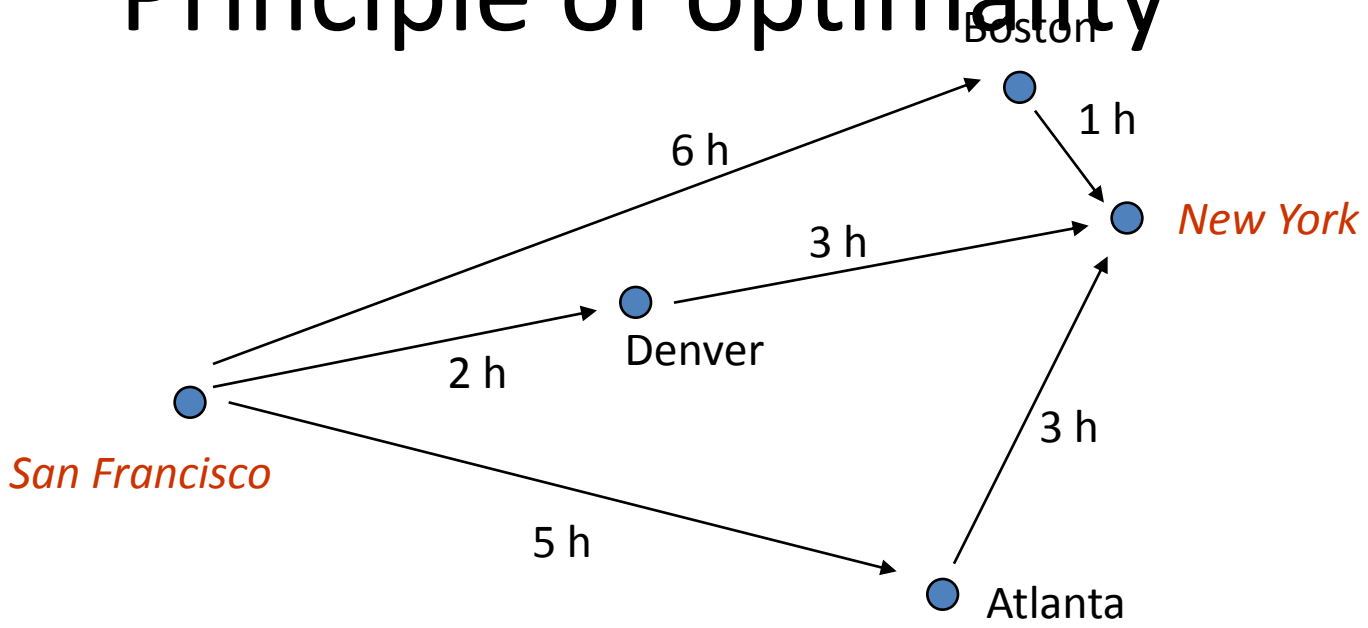
- Given the sequences and the edit transcripts, it is easy to find the alignment for the transcript.
- Alignment and edit transcript are equivalent. The transcript explicitly shows the mutational events and alignment displays the relationship between the strings.
- An alignment corresponding to the minimum cost score between the two strings is called an ***optimal alignment***.
- If the score is expressed in terms of number of substitution, insert or delete operations, then it is called a minimum edit distance alignment.
- In the similarity version of the problem, one seeks an alignment with maximum score where the score matrix gives the similarity values between the pairs of symbols.

Principle of optimality

- In some optimization problems...
- ...components of a globally optimal solution are themselves globally optimal
- Thus, we can optimize by recursively optimizing sub-problems

Courtesy : Bob Edgar, UC Berkeley

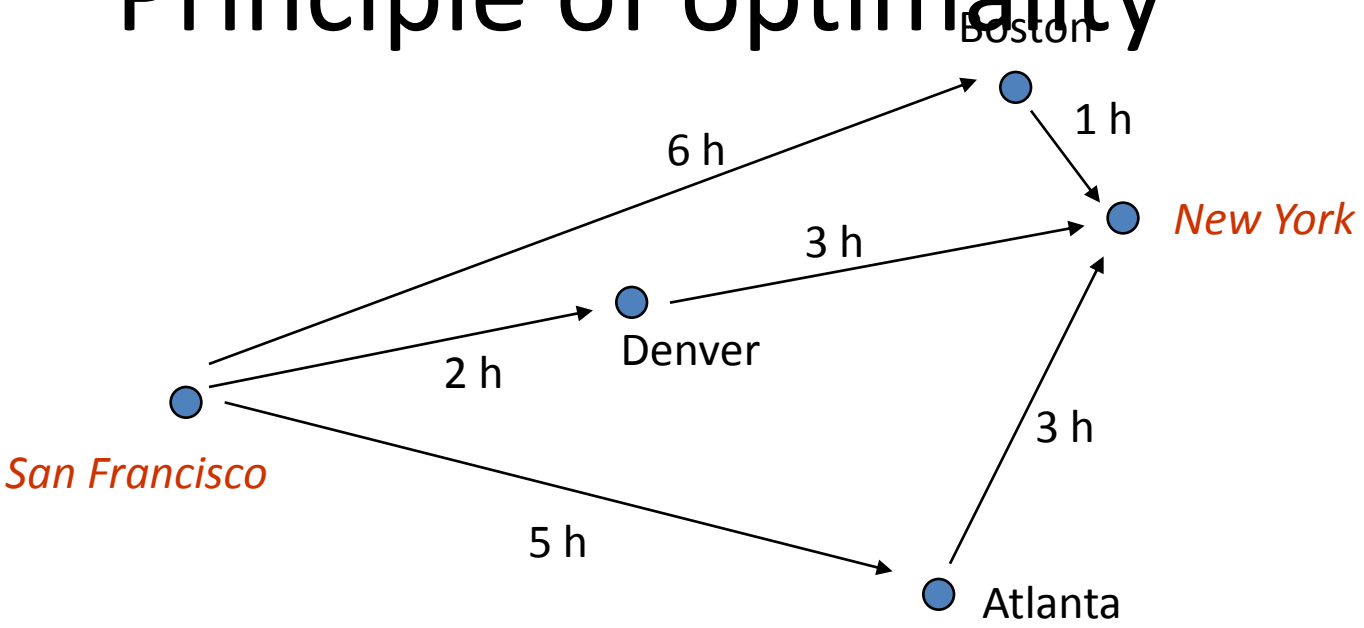
Principle of optimality



- Want fastest time San Francisco to NY, given:
  (1) You must fly via Denver (D), Boston (B) or Atlanta (A)
  (2) Fastest times from SF to D, B or A, and
  (3) Fastest times from D, B or A to NY.

Courtesy : Bob Edgar, UC Berkeley

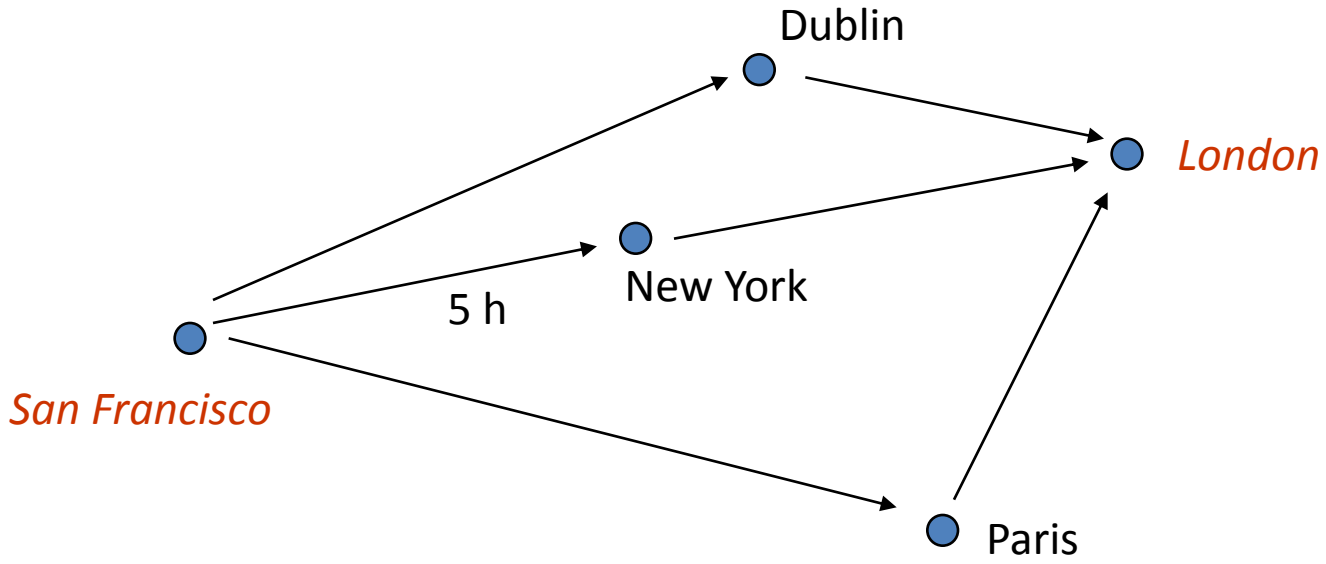
Principle of optimality



- Answer: find <u>minimum</u> of the <u>three possible routes</u>:
  - SF to B + B to NY
  - SF to D + D to NY
  - SF to A + A to NY
- = min (6 + 1, 2 + 3, 5 + 3) = min (7, 5, 8) = 5.

Courtesy : Bob Edgar, UC Berkeley

Principle of optimality



- Now want fastest time to London
- Must fly via New York, Dublin or Paris
- Doesn't matter how we get to NY, already know best time is 5h
- If we solve same problem for Dublin and Paris, can find the answer in the same way as for NY

Courtesy : Bob Edgar, UC Berkeley

Dynamic Programming

- Principle of optimality holds
- Solves simpler sub-problems
- Remember the results
- Use recursion to solve the next-biggest problem

Needleman-Wunsch Algorithm* : Dynamic Programming Formulation

*Needleman, S.B. and Wunsch, C.D. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol* **48**(3): 443-53.

Our problem is: given two sequences $v=a_1a_2\ldots a_n$ and $w=b_1b_2\ldots b_m$ and a scoring function δ, find an alignment denoting a set of "evolutionary" operations that converts the sequence *v* to *w* with minimum total cost. A graph theoretic formulation of the problem is of particular interest. The **edit graph** $G_{v,w}$ is an edge-labeled directed graph. The vertices of the graph are pairs (*i,j*) where $i \in [0,n]$ and $j \in [0,m]$. Imagine these vertices arranged in a *n*+1 by *m*+1 grid, as shown in the next slide for the sequences *v=ATGTTAT* and *w=ATCGTAC.* The vertex at locations (0,0) and (*n,m*) are called **source** and **sink** nodes. The graph has the following edge set:

Edit Graph (contd.)

1. If $i \in [1,n]$ and $j \in [0,m]$ then, a deletion edge (i-1,j) →(i,j) denoted $(a_i,-)$
2. 1. If $i \in [0,n]$ and $j \in [1,m]$ then an insertion edge (i,j-1) →(i,j) denoted $(-,b_j)$
3. 1. If $i \in [1,n]$ and $j \in [1,m]$ then , a substitution edge (i-1,j-1) →(i,j) denoted (a_i,b_j)

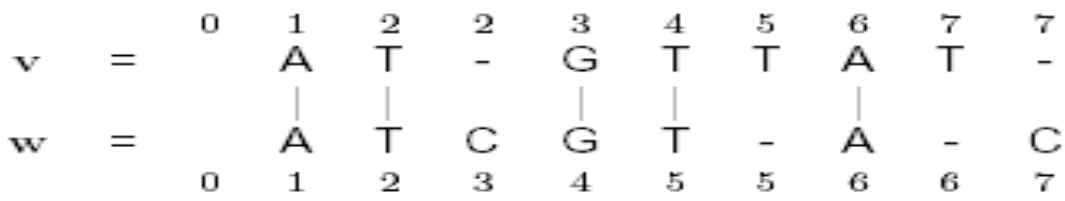
The next slide shows the edit graph for an example.
The deletion edges are vertical and the insertion edges are horizontal.

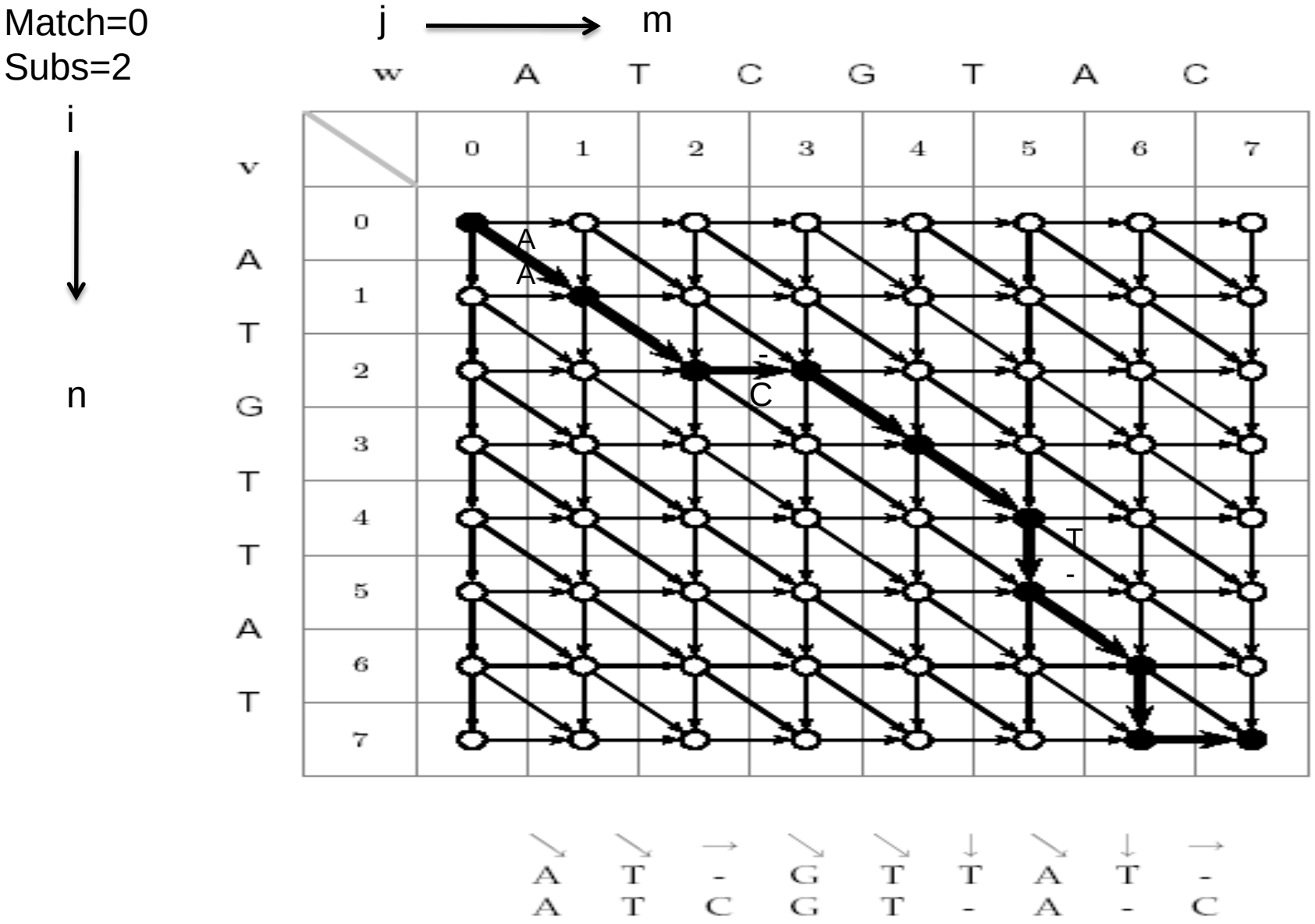
Let V_i denote the prefix $(a_1a_2..a_i)$ of v $(0 \le i \le n)$. For i=0, $V_i = \epsilon$ (null sequence).Similarly, let W_j denote the jth prefix of w, that is, $(b_1b_2\ldots b_j)$. For j=0, $W_j = \epsilon$ (null sequence). A path in the edit graph spells the alignment obtained by concatenating the edge labels in the path. Thus, a path from the vertex (0,0) to the vertex (i,j) spells an alignment of V_i and W_j. A different path spells a different alignment. This can be proved as follows: An alignment between V_i and W_j must end up with one of the aligned pairs $(a_i,-)$, $(-,b_j)$ or (a_i,b_j). Thus the alignment between V_i and W_j must be one of :

1. Alignment between V_{i-1} and W_j concatenated with $(a_i,-)$
2. Alignment between V_i and W_{j-1} concatenated with $(-,b_j)$
3. Alignment between V_{i-1} and W_{j-1} concatenated with (a_i,b_j)

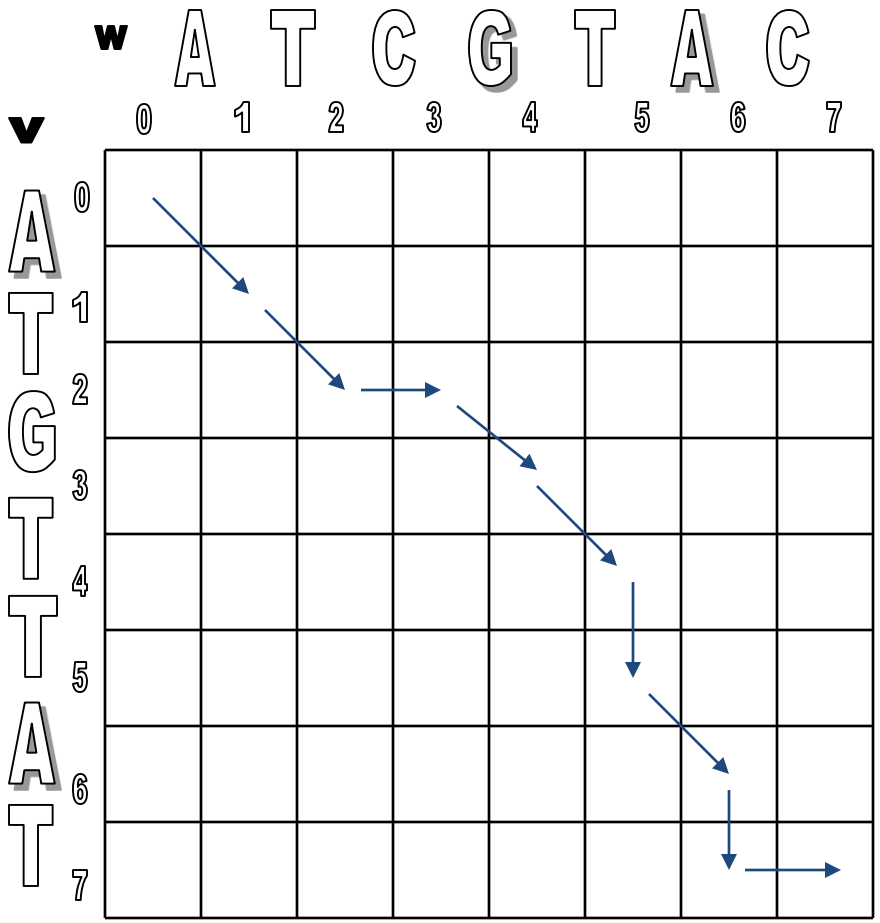
Thus there is a one-to-one correspondence between paths in the edit graph from vertex (0,0) to vertex (n,m) and alignments of v and w.

Cost Model:
Indel=1
Match=0
Subs=2

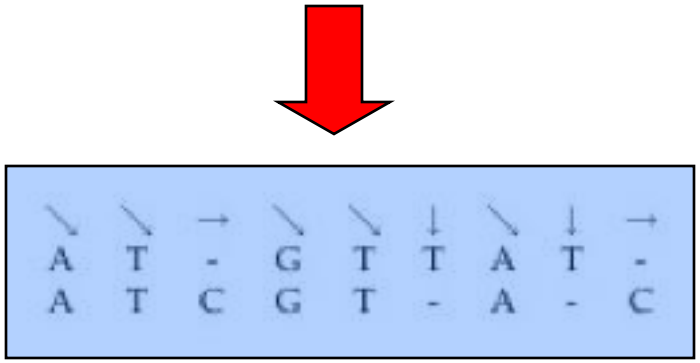




Alignment as a Path in the Edit Graph



Every path in the edit graph corresponds to an alignment:



Computation of Optimal Alignments

The goal is to compute an optimal alignment(s) of v and w. We assign weights $\delta(a,b)$ to an edge if it is labeled (a,b). Here, either *'a'* or *'b'* could be the character '-'. Thus, the problem reduces to finding minimum weight path(s) from vertex (0,0) to vertex (n,m) in the edit graph as weighted by δ. The edit graph $G_{v,w}$ is a directed acyclic graph and it needs to be traversed in *topological order* to obtain the minimum weight path in a single pass. The possible orders are row, column or diagonal orders. We will use row order sweeping left-to-right within a row. The total cost $C(i,j)$ of the minimum weight path from vertex (0,0) to the vertex (i,j) is given by equations given in the next slide.

Computation of Optimal Alignments

$C(0,0) \leftarrow 0$

for $j = 1$ to m do

$C(0, j) \leftarrow C(0, j-1) + \delta(-, b_j)$ /* insert from w */

for $i = 1$ to n do

{ $C(i,0) \leftarrow C(i-1,0) + \delta(a_i, 0)$ /* delete from v */

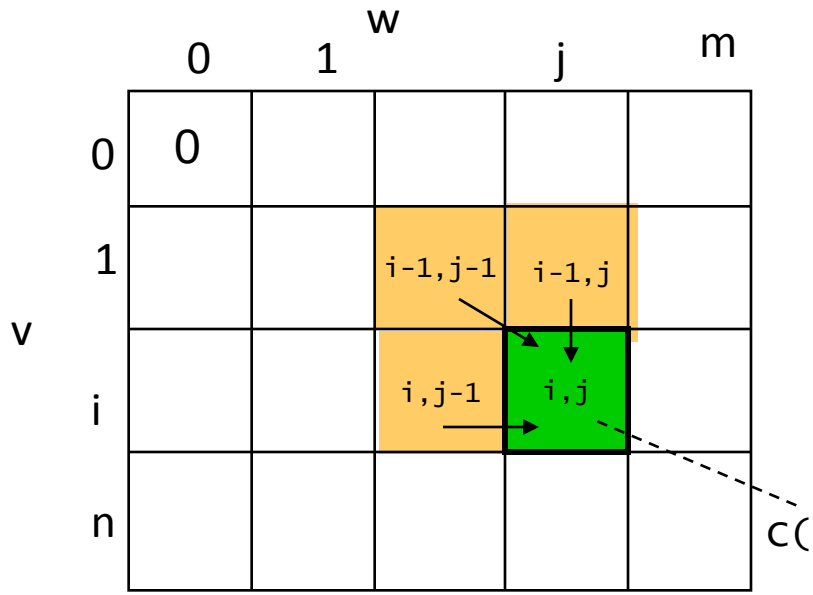
for $j = 1$ to m do

$C(i, j) \leftarrow \min\{ C(i, j-1) + \delta(-, w_j),\ C(i-1, j) + \delta(v_i, -),\ C(i-1, j-1) + \delta(a_i, b_j)\}$

}

write "difference score is" $C(n, m)$

The Recurrence Relation

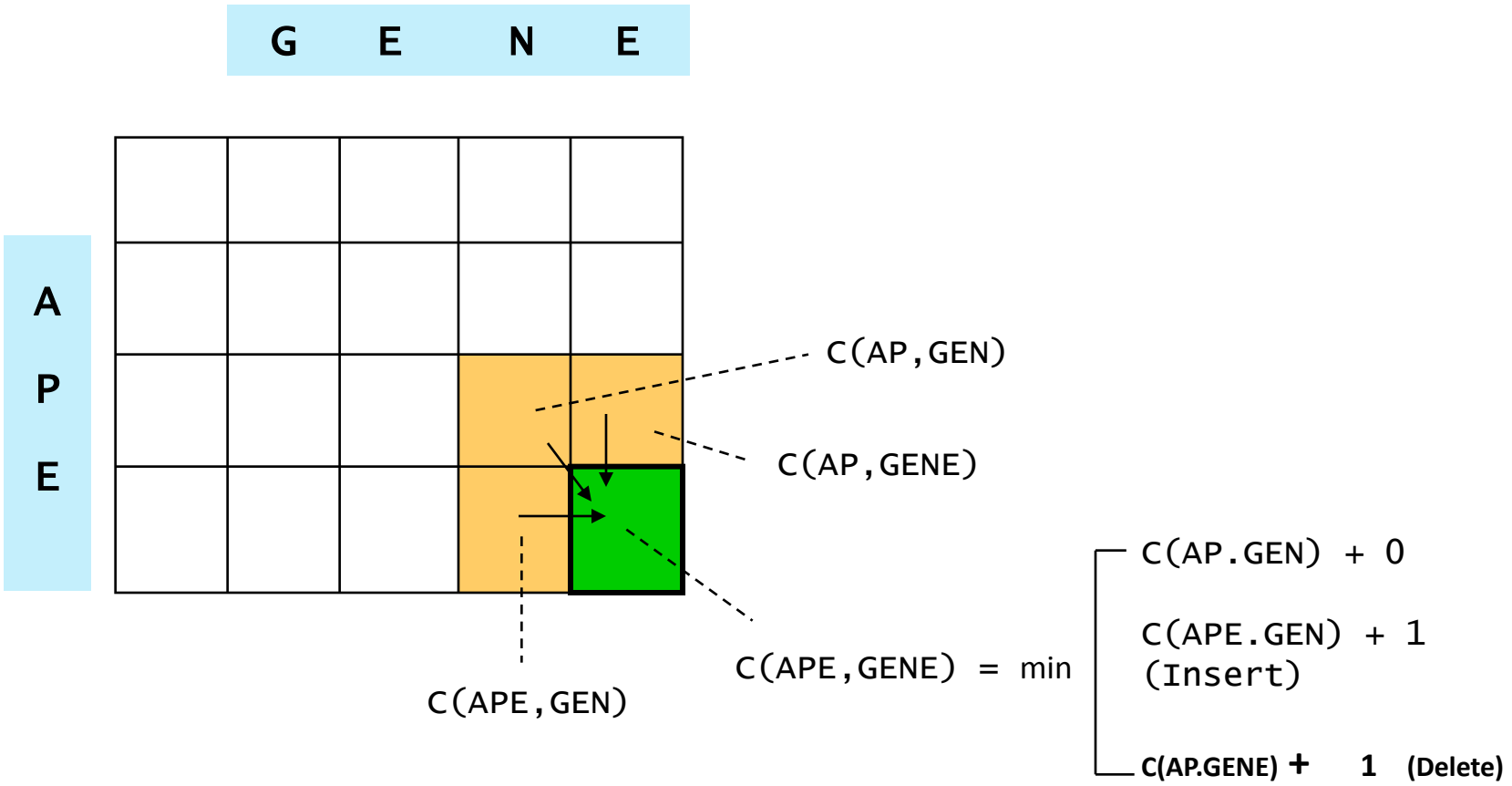


C(0,0)=0 when i=0 and j=0
C(0,j)=C(0,j-1)+ δ(-.w_j) when i=0, j>0
C(i,0)=C(i-1,0)+ δ(v_i,-)when i>0, j=0

C(i,j) = min
C(i-1,j) + δ(v_i,-) Delete
C(i,j-1) + δ(-,w_j) Insert
C(i-1,j-1) + δ(v_i,w_j)

Consider a minimum cost edit transcript for *C*(*i,j*). If the last operation of this transcript is an **insert** I operation, then the alignment must have been at this point (-, w_j) corresponding to the **horizonta**l arrow in the matrix. If the last operation of this transcript is a **delete** operation, then the alignment must have been at this point (v_i, -) corresponding to the **vertical** arrow in the matrix. Otherwise, the computation must have taken the diagonal arrow in the matrix which might correspond to either a match or a replacement of v_i by w_j.

Edit distance matrix M



Recursive Procedure

- The recursive procedure is a top-down approach theoretically. That is, the computation starts at the lower rightmost point. In practical implementation, it might need an **exponential** number of calls.
- We have given a bottom-up tabular computation which is more efficient.
- To compute the value at any point in the matrix, it is sufficient if we know the minimum cost alignments of its **north, north-west and west neighbors** and the pairs of characters from the two sequences under consideration.
- After computing the 0th row and the 0th column of the matrix (the minimum edit distance is simply the index of the row or column if we are only interested in edit distance), we compute the rest of the matrix one row at a time consecutively with increasing row indices or one column at a time consecutively with increasing column indices.

Time Complexity

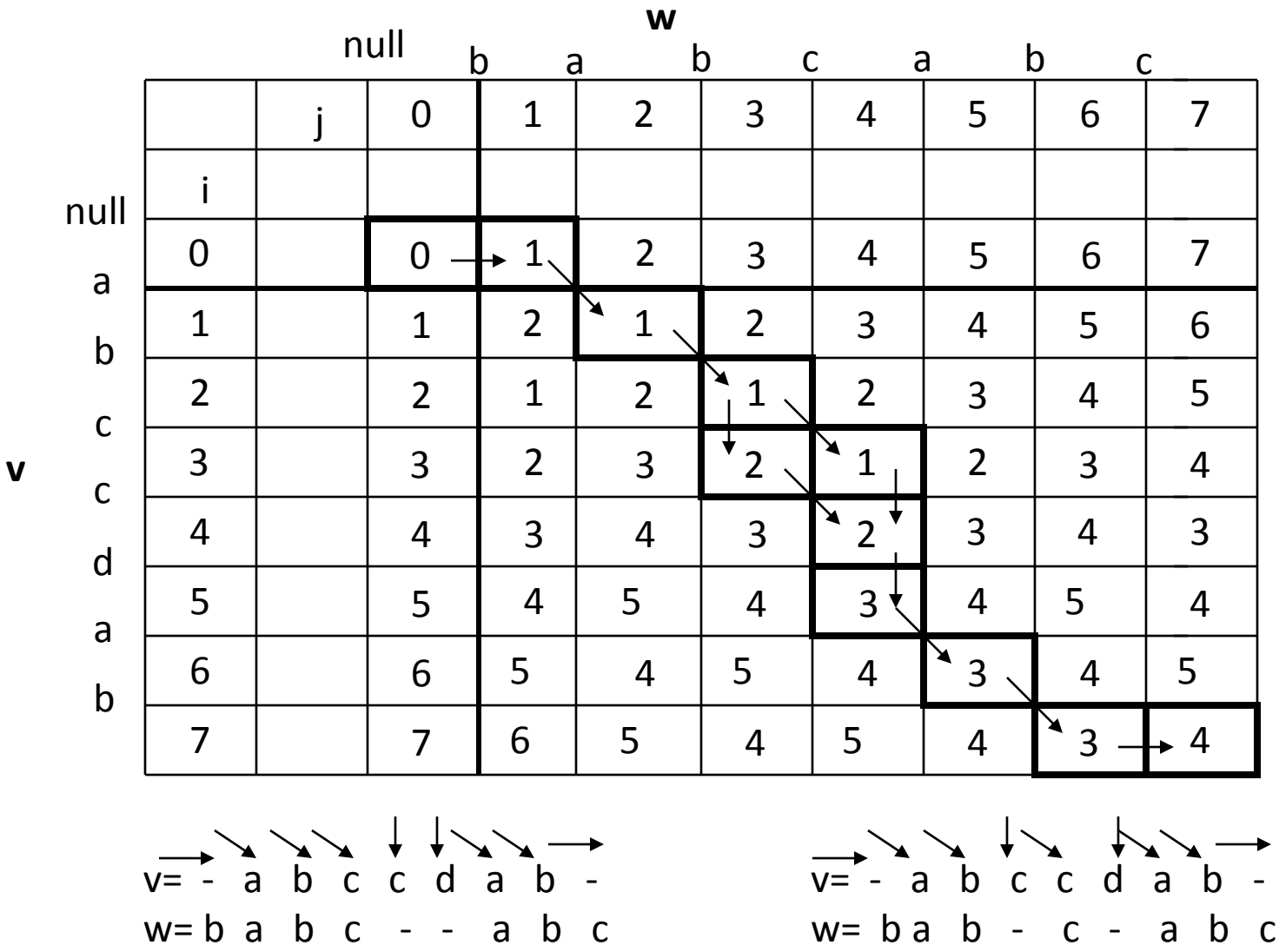
- **Theorem**: The dynamic programming algorithm computes a minimum cost alignment in time $O(nm)$.
  - Proof. The algorithm needs an $(n+1)(m+1)$ table to be computed. Any particular entry in the table involves three additions, one character comparison operation and one three-way minimum value computation, all requiring $O(1)$ time. Hence the total time is $O(nm)$.

Time Complexity

- **Theorem**: Once the dynamic programming table with pointers has been computed, an optimal cost alignment can be found taking $O(n+m)$ time.
  - Proof. During the construction of the table, the back pointers to neighboring cells having minimum cost values can be set up taking $O(nm)$ storage and time. Then a directed path of back pointers originating from (n,m) to (0,0) , called a *trace,* can be constructed taking only $O(n+m)$ time since at each step the path must extend to north, west or north-west.

Time Complexity

- **Theorem**: Every trace from (*n,m*) to (0,0) corresponds to an optimal alignment in one-to-one fashion.
  - Proof: Every point in the trace represents a minimum cost alignment from (0,0) to that point.



An example to illustrate edit distance Computation

Indel cost=1
Match=0
Subs.=2

Note, the insertion from **w** is denoted by a horizontal arrow, a deletion from **v** by a vertical arrow and a replacement or match is denoted by a diagonal arrow. For this example, the there are two possible alignments with 'gaps' created for indel operations.

Another Example

	ϵ	A	T	C	C	G	A	T
ϵ	0	1	2	3	4	5	6	7
T	0	0	←1	←2	←3	←4	←5	←6	←7
A	1	↑1	↖1	↖1	←2	←3	←4	←5	←↖6
T	2	↑2	↖1	←↖↑2	↖2	←3	←4	↖4	←5
C	3	↑3	↑2	↖1	←2	←↖3	←↖4	←↖↑5	↖4
A	4	↑4	↑3	↑2	↖1	←2	←3	←4	←↑5
T	5	↑5	↖↑4	↑3	↑2	↖2	←↖3	↖3	←4
C	6	↑6	↑5	↖↑4	↖↑3	↖↑3	↖3	←↖↑4	↖3
	7	↑7	↑6	↑5	↖↑4	↖3	←↖↑4	↖4	↑4

Path

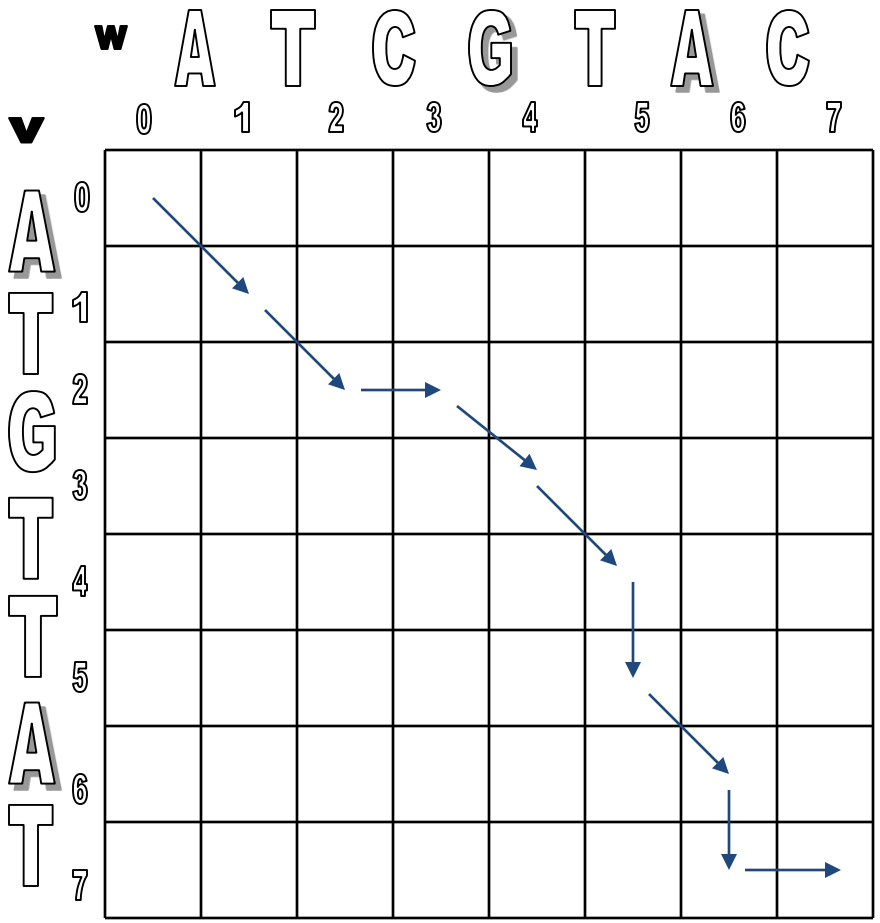
d(i,j)			A	T	C	C	G	A	T
		0	1	2	3	4	5	6	7
	0	**0**							
T	1	**↑1**							
A	2		**↖1**						
T	3			**↖1**					
C	4				**↖1**	**←2**	**←3**		
A	5							**↖3**	
T	6								**↖3**
C	7								**↑4**

Alignment:

v T A T C _ _ A T C

w _ A T C C G A T _

Similarity as a Path in the Edit Graph: 2-dimensional coordinates

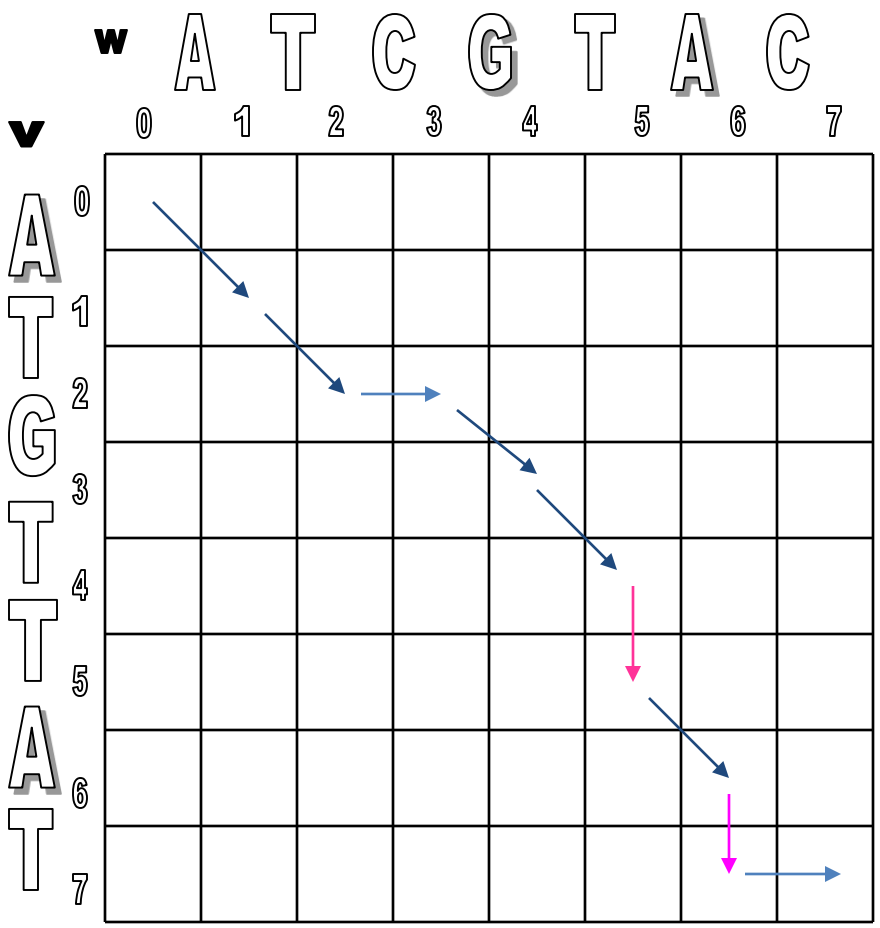


0 1 2 2 3 4 5 6 7 7
A T _ G T T A T _
A T C G T _ A _ C
0 1 2 3 4 5 5 6 6 7

- Corresponding path -

(0,0) , (1,1) , (2,2), (2,3), (3,4), (4,5), (5,5), (6,6), (7,6), (7,7)

Similarity using Alignments

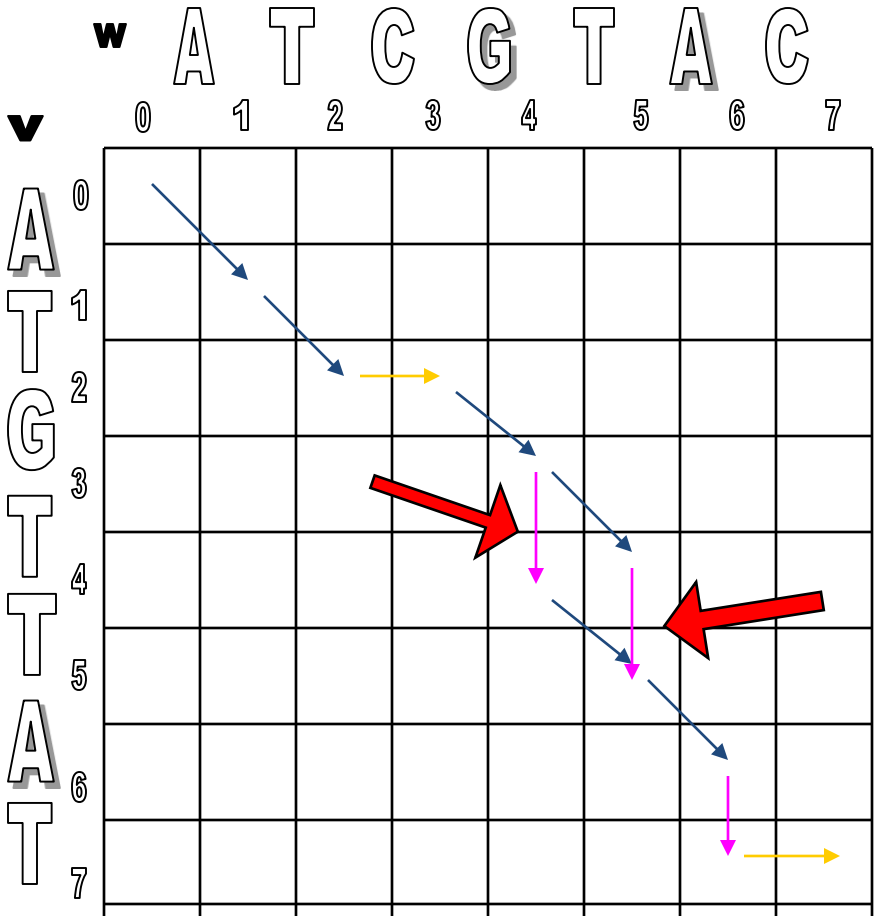


↓ and → represent indels in **v** and **w** with score 0.

↘ represent matches with score 1.

- The similarity score of the alignment path is 5.

Two Paths in the Edit Graph having same similarity score



Path 1

0122345677
v= AT_GTTAT_
w= ATCGT_A_C
0123455667

Path 2

0122345677
v= AT_GTTAT_
w= ATCG_TA_C
0123445667

Operation-Weight Alignment

- With arbitrary weights, the solution will correspond to a minimum weighted path between two points. This allows us to define more complex alignment problems between two strings.
- We can assign weights based on operation (*I,D,R*, or *M*), called ***operation-weight alignment.*** The minimum cost solution that we presented based on cost table is also known ***alphabet-weight alignment***.