Assignment - IV

Sequence Alignment

Name: Gowlapalli Rohit

Roll No: 2021101113

(1) Given two nucleotide sequences

GGCTGCAACTAGCTC

GGGTAAGCTTGC

and the transition - transversion scoring matrix (expression in similarity)

	A	C	G	T
A	4	-1	1	-1
C	-1	4	-1	1
G	1	-1	4	-1
T	-1	1	-1	4

and gap-penalty = -3

Global alignment:

Boundary conditions: $F(i,0) = -id \;\forall\; i \in [0,n]$

$F(0,j) = -jd \;\forall\; j \in [0,m]$

→ A pointer is kept in each cell back to the cell from which its $F(i,j)$ was derived

The best score upto (i,j) will be the largest of these 3 options

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + s(x_i, y_j) \\ F(i-1,j) - d \\ F(i,j-1) - d \end{cases}$$

d = gap-penalty

$F(n,m)$ gives the best score for global alignment of $x_{1\ldots n}, y_{1\ldots m}$

After using Needleman-Wunsch Algorithm for the above sequences

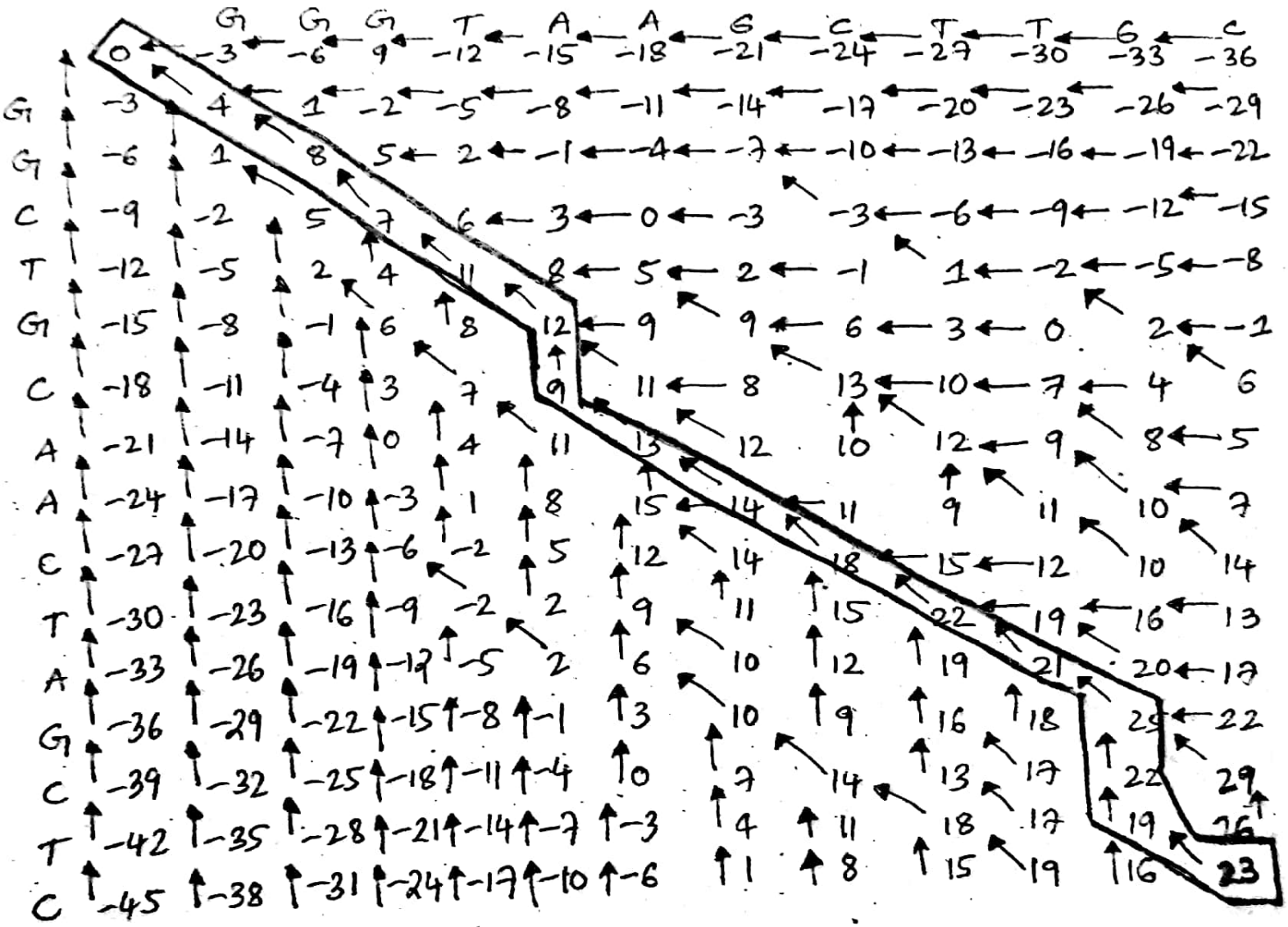
Final similarity score = 23

Best Alignment:

GGCTGCAACTAGCTC

GGGTA_AGCTTG--C

```python
import numpy as np
def global_alignment(seq1, seq2, scoring_matrix, gap_penalty):
    dp_matrix = np.zeros((len(seq1) + 1, len(seq2) + 1))
    for i in range(1, len(seq1) + 1):
        dp_matrix[i][0] = gap_penalty * i
    for j in range(1, len(seq2) + 1):
        dp_matrix[0][j] = gap_penalty * j
    for i in range(1, len(seq1) + 1):
        for j in range(1, len(seq2) + 1):
            match_score = scoring_matrix[seq1[i-1]][seq2[j-1]]
            dp_matrix[i][j] = max(dp_matrix[i-1][j-1] + match_score,dp_matrix[i-1][j] + gap_penalty,dp_matrix[i][j-1] + gap_penalty)
    align1 = ""
    align2 = ""
    i = len(seq1)
    j = len(seq2)
    while i > 0 or j > 0:
        if i > 0 and j > 0 and dp_matrix[i][j] == dp_matrix[i-1][j-1] + scoring_matrix[seq1[i-1]][seq2[j-1]]:
            align1 = seq1[i-1] + align1
            align2 = seq2[j-1] + align2
            i -= 1
            j -= 1
        elif i > 0 and dp_matrix[i][j] == dp_matrix[i-1][j] + gap_penalty:
            align1 = seq1[i-1] + align1
            align2 = "-" + align2
            i -= 1
        else:
            align1 = "-" + align1
            align2 = seq2[j-1] + align2
            j -= 1
    return align1, align2, dp_matrix[-1][-1], dp_matrix
seq1 = "GGCTGCAACTAGCTC"
seq2 = "GGGTAAGCTTGC"
scoring_matrix = {"A": {"A": 4, "T": -1, "C": -1, "G": 1},"T": {"A": -1, "T": 4, "C": 1, "G": -1},
    "C": {"A": -1, "T": 1, "C": 4, "G": -1},"G": {"A": 1, "T": -1, "C": -1, "G": 4}}
gap_penalty = -3
align1,align2, score, dp_matrix = global_alignment(seq1, seq2, scoring_matrix, gap_penalty)
```



GGCTGCAACTAGCTC

GGGTA_AGCTTG__C

Local Alignment:

Boundary conditions: $F(i,0) = 0$ and $F(0,j) = 0$

$\forall i \in [0,n]$ $\forall j \in [0,m]$

$$F(i,j) = \max \begin{cases} 0 \\ F(i-1, j-1) + S(x_i, y_j) \\ F(i-1, j) - d \\ F(i, j-1) - d \end{cases}$$

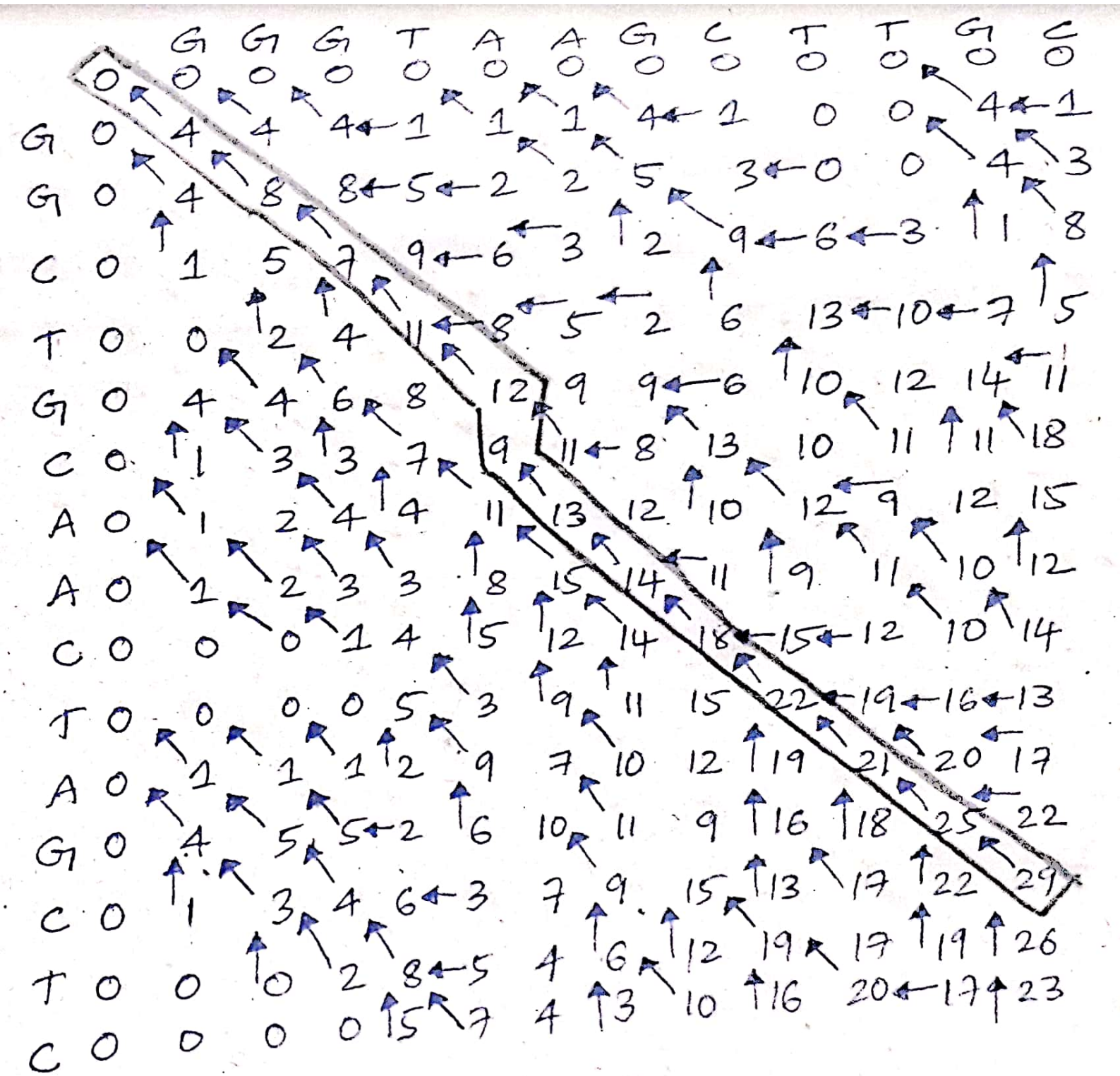
After using Smith-Waterman Algorithm, for using it to the above sequences,

Similarity score = 29

Best Alignment:

GGCTGCAACTAGC
GGGTA-AGCTTGC

```python
import numpy as np
def local_alignment(seq1, seq2, scoring_matrix, gap_penalty):
    dp_matrix = np.zeros((len(seq1) + 1, len(seq2) + 1))
    for i in range(1, len(seq1) + 1):
        for j in range(1, len(seq2) + 1):
            match_score = scoring_matrix[seq1[i-1]][seq2[j-1]]
            dp_matrix[i][j] = max(0,dp_matrix[i-1][j-1] + match_score,dp_matrix[i-1][j] + gap_penalty,dp_matrix[i][j-1] + gap_penalty)
    max_score = 0
    max_i = 0
    max_j = 0
    for i in range(len(seq1) + 1):
        for j in range(len(seq2) + 1):
            if dp_matrix[i][j] > max_score:
                max_score = dp_matrix[i][j]
                max_i = i
                max_j = j
    align1 = ""
    align2 = ""
    i = max_i
    j = max_j
    while i > 0 and j > 0 and dp_matrix[i][j] > 0:
        if dp_matrix[i][j] == dp_matrix[i-1][j-1] + scoring_matrix[seq1[i-1]][seq2[j-1]]:
            align1 = seq1[i-1] + align1
            align2 = seq2[j-1] + align2
            i -= 1
            j -= 1
        elif dp_matrix[i][j] == dp_matrix[i-1][j] + gap_penalty:
            align1 = seq1[i-1] + align1
            align2 = "-" + align2
            i -= 1
        else:
            align1 = "-" + align1
            align2 = seq2[j-1] + align2
            j -= 1
    return align1, align2, max_score, dp_matrix
seq1 = "GGCTGCAACTAGCTC"
seq2 = "GGGTAAGCTTGC"
scoring_matrix = {
    "A": {"A": 4, "T": -1, "C": -1, "G": 1}, "T": {"A": -1, "T": 4, "C": 1, "G": -1},
    "C": {"A": -1, "T": 1, "C": 4, "G": -1},"G": {"A": 1, "T": -1, "C": -1, "G": 4}}
gap_penalty = -3
align1, align2, score, dp_matrix = local_alignment(seq1, seq2, scoring_matrix, gap_penalty)
```



GGCTGCAACTAGC

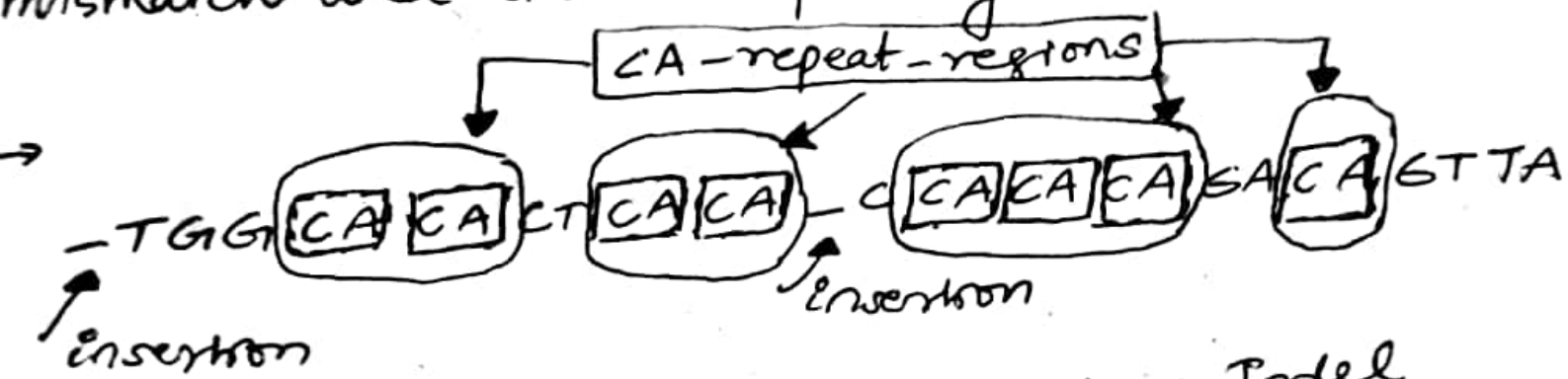
GGGTA-AGCTTGC

(2) Tandem repeats are a common type of repetitive DNA sequence found in many organisms, including humans. They occur when a pattern of nucleotides is repeated and the repetitions are directly adjacent to each other. Tandem repeats can be found in various regions of the genome, such as introns, exons & non-coding regions

Given the sequence

TGGCACACTCACACCACACAGACAGTTA

to find the dinucleotide CA repeat region using the scoring-scheme (1, 0, -1) for match, mismatch and indel respectively



−1 → Indel

match → 1

mismatch → 0

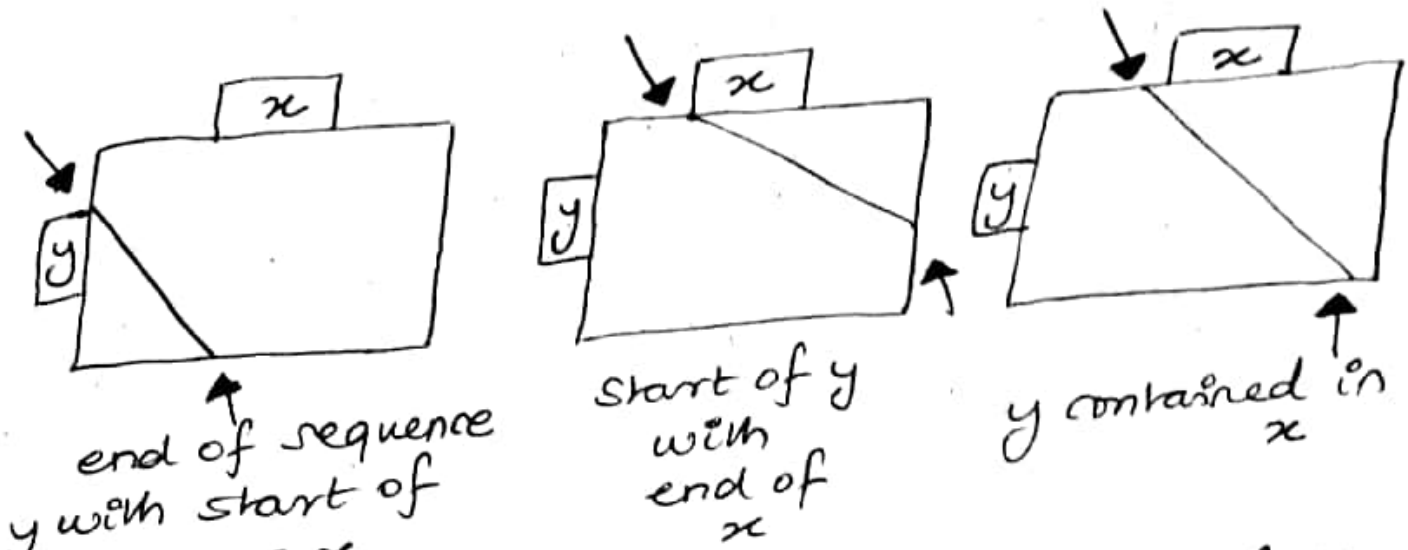
Score = −1+0+1
+1+0+1+1
+−1+1+1+1
+0+1+0+0

= 8 − 2 = 6

Score = 6

(3) Overlap matches is a special case of global alignment that does not penalize overhanging ends - also called semi-global alignment.

i.e, an algorithm for a match that may start on the Left / Top border of the matrix, and end on the right / bottom border



Initialisation equations: (same as for local alignment)

$$F(0,0) = 0$$
$$F(i,0) = 0 \quad i = 1, \ldots n$$
$$F(0,j) = 0 \quad j = 1, \ldots m$$

Recurrence relations (same as for global alignment)

$$F(i,j) = \max \begin{cases} F(i-1, j-1) + S(x_i, y_j) \\ F(i-1, j) - d \\ F(i, j-1) - d \end{cases}$$

F_{max} = the maximum value on the bottom border (i, m), $i = 1, \ldots, n$ or the right border (n, j) $j = 1, \ldots m$

→ Traceback starts from Fmax and continues until the top (i,0) or left (0,j) edge is reached

→ This problem arises in various applications such as genome assembly, where the goal is to piece together the fragments of a genome obtained from sequencing reads. The overlap b/w adjacent reads needs to be identified to correctly assemble the genome

→ Overlapping regions occur when one sequence is part of other (or) when both sequences have common overlapping regions

For Ex: Comparing fragments of DNA Sequences to each other (or) to a large chromosomal sequences

④ Affine gap cost is given by

$$\gamma(g) = -d - (g-1)e$$

The advantage of using affine gap scores is that they provide a more biologically realistic model for the introduction & extension of gaps in sequence alignments

In traditional gap penalties, a single fixed penalty is assigned for each gap introduced into the alignment. However in reality, gaps in biological sequences may occur in clusters, and the cost of introducing a gap and extending it can vary. For example, a deletion of 1 nucleotide followed by the deletion of another nucleotide is more likely than the deletion of 2 non-consecutive nucleotides. Therefore, affine gap penalties are introduced to account for this phenomenon

Affine gap penalties are compared of 2 seperate penalties: a gap-opening penalty and a gap-extension penalty. The gap-opening penalty is the cost of introducing a gap, while the gap-extension penalty is the cost of adding nucleotides to an existing gap. The gap-extension penalty is usually smaller than the gap-opening penalty, reflecting the fact that once a gap is introduced, the cost of extending it is lower

Using affine gap scores can lead to more accurate & biologically relevant sequence alignments by providing a more realistic model for the introduction & extension of

gaps in sequences. This is particularly important for accurately aligning sequences with long-gap regions or for detecting sequences similarities in sequences with variable-length gaps

d = gap open penalty
e = gap extension penalty

→ Affine score assumes that consecutive deletions and insertions & hence should be penalized

→ Affine score assumes that the above are single mutation events as opposed to multiple insertions/deletions and hence should be penalized less

→ These also provide most sensitive sequence matching methods

→ While aligning sequences introducing gaps in the sequences can allow an alignment algorithm to match more terms than a gap-less alignment

(5) The time and space complexity of dynamic programming algorithms for sequence comparison depends on the length of sequences being compared

→ It is of the order – $O(nm)$ → both the Time and Space complexity where n, m are length of two sequences

→ For global alignment, local alignment and Affine-gap penalty algorithms have a time-complexity of- $O(nm)$ and space complexity of the order of $O(nm)$

→ Not feasible for comparing complete genomes or chromosomes ~ a few Mbs long

→ In database search, a query sequence of length n is searched a database of size ~ few Gbs → Time-complexity is an issue in this case

↘ space-complexity needs to be addressed

When the sequences being compared are short, time & space complexity may not be an issue. However, when the sequences are very long, the time & space requirements of DP Algorithms can become a significant problem. This is especially true for genome-scale comparisions, when sizes of the sequences can be in billions of base pairs

↘ seed and extend (or) k-mer approaches are used in such cases

→ DP algorithms although very sensitive, are not the fastest

→ Protein database contain ~100M residues, (In many cases, speed is the issue ↳ database searches)

this requires ~10^{11} matrix cells to be evaluated to search complete database for a query of length 1000; DNA Dbs are even larger

At 10M matrix cells per second this would be 10^4 secs or ~3hrs for a single-search

BLAST	PSI-BLAST
(1) It is a basic Local alignment search tool using similarity-matrix based search in 1/more databases, for sequences similar to query sequences of any type	(1) It is a position specific version of iterative BLAST. It iteratively searches in 1/more databases for sequences similar to 1/more query sequences
(2) Uses similar pairs instead of identical pairs Extension is without gaps	(2) Allows detection of Homologues in the range of 15% - 25% sequence identity levels
(3) Uses scoring matrix to score aligned pairs. Only those pairs above a score of threshold are considered for extension	(3) Constructs scoring matrices by multiple alignment of hits obtained
(4) BLAST uses similarity scoring matrices	(4) PSI-BLAST uses position specific scoring matrices derived at at every iteration
(5) Uncapped extension of HSP's with scores > T Identifies maximal segment pairs. Extensions continues until the score drops below a threshold drop-off from the maximum score encountered Highest scoring segment pair, MSP is identified	(5) Searches the database with new scoring matrix for every iteration, Iteration until convergence is reached The idea of constructing a scoring matrix from the hits is that the new scoring matrix is tailor-made to find sequences similar to the query

→ PSI-BLAST builds a Position-Specific scoring matrix (PSSM) from the results of the previous BLAST iteration which is then used to perform a new search against the base

→ In contrast the blastp program performs a standard BLAST search without iteration, using a matrix of pairwise scores to compare the query sequence against sequences in a protein database

→ The key-difference is that PSI-BLAST results in a more sensitive search that can detect more distant homologs. Additionally, PSI BLAST allows for the identification of conserved regions & domains in the query sequence, and can be used to detect remote homologs that may not be identified by standard BLAST searches

→ blastp compares a protein query to a protein-database whereas PSI-BLAST allows the user to build a PSSM using the results of the first BLASTP run

(7) In BLAST database search algorithm, the match/mismatch ratio is chosen to be small for highly conserved sequences, while it is large for divergent sequences

match (M) / mismatch (N) → Relative magnitudes of M & N determines the No. of nucleic acid PAM's (point accepted mutations per 100 residues) for which they are most sensitive at finding homologs

the (absolute) reward/penalty ratio should be increased as one looks at divergent sequences

Reasons

(1) Highly conserved sequences have a lower number of substitutions and are more similar to each other. Therefore, using a small match/mismatch ratio helps to emphasize the conservation of the sequences & increase the sensitivity of algorithm to detect matches

(2) Divergent sequences have a higher number of substitutions and are more dissimilar to each other. Therefore, using a large match/mismatch ratio helps to emphasize the differences b/w the sequences and increase the specificity of the algorithm to avoid false matches

(3) It is because a small ratio is more sensitive to small differences, while a larger ratio is more tolerant to large differences

(4) The choice of match/mismatch ratio also depends on the specific application & the goal of analysis.

(8) Given that in BLOSUM62 matrix, a conserved Tryptophan has score $s(W,W) = 11$, but conserved Leucine position has score $s(L,L) = 4$

→ They reflect the differences in the frequency of occurence of the 2 amino acids in the protein sequences used to construct the matrix

→ Tryphophan is a less common amino acid than Leucine, so it is more likely that a substitution at a conserved Tryphophan position would be deleterious for protein function. Therefore, the penalty for substituting Tryphophan with any other amino acid is higher in the BLOSUM62 matrix

→ On the other hand, Leucine is more common in protein sequences, so it is more likely that a substitution at a conserved Leucine position would be tolerated. Hence, the penalty for substituting Leucine with any other amino acid is lower in BLOSUM62 matrix

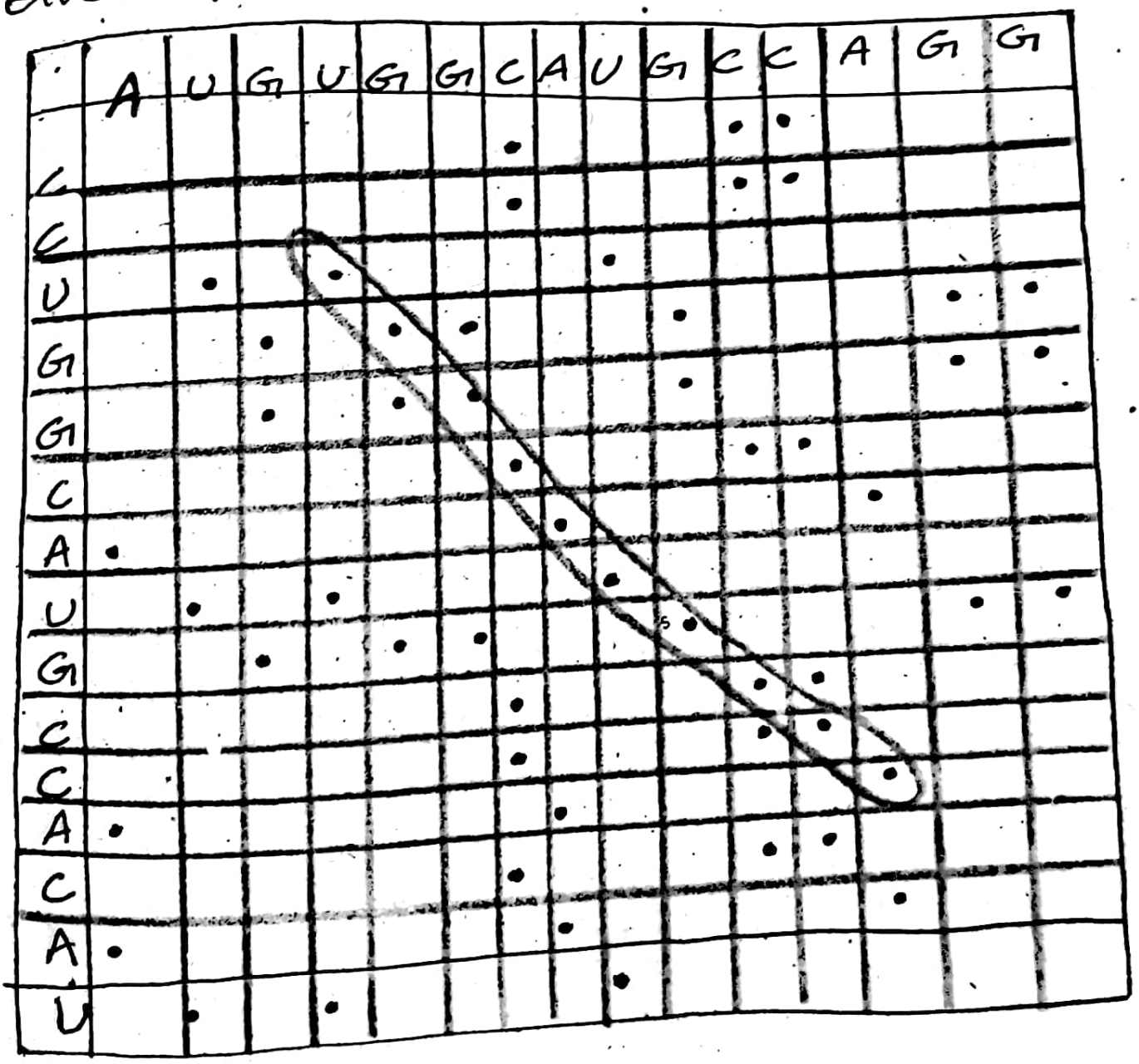
→ The scores are different for identical residues depending on the degree to which that residue is found to be highly conserved in most proteins

(9)

Method-1 : Sequence is listed in 5' to 3' direction using the horizontal axis and it's complementary sequence is listed along the vertical axis, also in the 5' to 3' direction

Matrix is then stored for identities

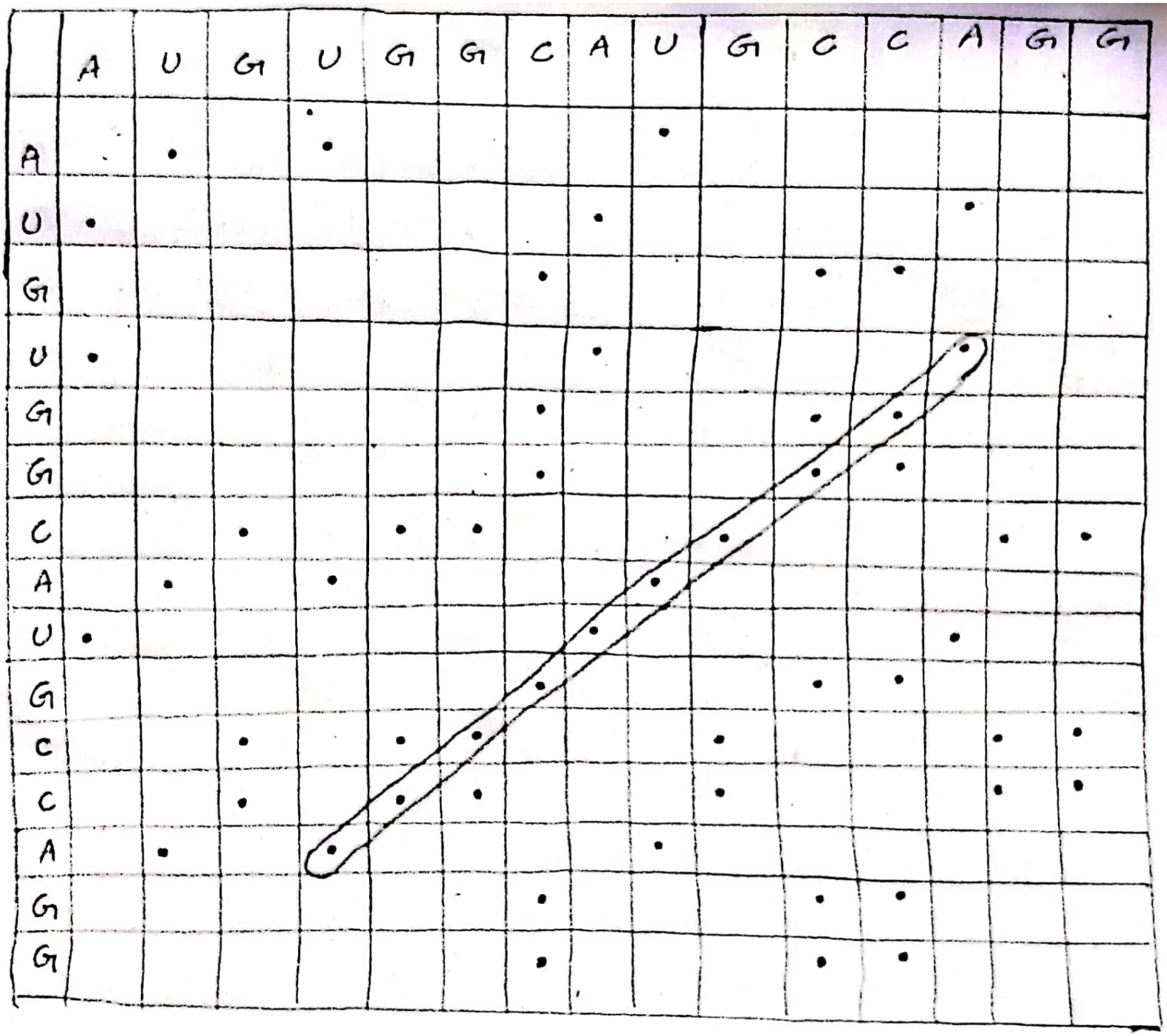
Given sequence = AUGUGGCAUGCCAGG



Method-2

Alternative approach - list the RNA sequence along the horizontal axis and also along the vertical axis

- Score matches of complementary bases G/C, A/U and G/U instead of identities
- Diagonals indicating complementary region will go from upper right to lower left in this matrix



The self-complementary region in the above sequence is given by UGGCAUGCCA

(conferred from dotplots)