Implementation of the Smith-Waterman Algorithm with Affine Gap Penalties for Identifying Conserved Protein Domains

Varel Tiara - 13523008
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: vareltiara@gmail.com , 13523008@std.stei.itb.ac.id

***Abstract*—Sequence alignment is a foundational technique in bioinformatics for inferring biological relationships. The Smith-Waterman algorithm is a key method for local alignment, but its accuracy depends critically on the gap penalty model. This paper implements and comparatively analyzes the Smith-Waterman algorithm using two models: a simple linear penalty and the more complex affine gap penalty. The affine model, with separate penalties for opening and extending a gap, is considered more biologically realistic. To test this hypothesis, the algorithms were applied to a specific case study: the identification of the conserved DNA-binding domain of the human p53 tumor suppressor protein (UniProt: P04637) within a homologous protein from Drosophila melanogaster (UniProt: Q9N6D8). The results demonstrate that while both models can identify similarity, the affine gap penalty model produces a more coherent and structurally sound alignment, consolidating gaps into larger, contiguous blocks that are more consistent with evolutionary events affecting protein loop regions. This alignment strengthens the biological hypothesis for the conservation of the p53 domain's core structure. Ultimately, this work demonstrates the choice of a computational model is not a mere technical detail, but a critical step that directly shapes the validity of biological hypotheses.**

Keywords*—Dynamic Programming, Smith-Waterman Algorithm, Local Sequence Alignment, Affine Gap Penalty, Bioinformatics, Protein Domain, p53, BLOSUM62

I. Introduction

The field of bioinformatics has emerged as an indispensable pillar of modern biology, facilitating discoveries in genomics, proteomics, and evolutionary biology through the power of computational analysis. At the heart of this new scientific paradigm lies sequence alignment, a computational process for comparing biological sequences to identify regions of similarity. This process is not merely a pattern-matching exercise. It is the primary method through which researchers probe the evolutionary, structural, and functional relationships encoded within the molecules of life. The intellectual framework that justifies this approach is the sequence-structure-function paradigm, a central tenet of molecular biology which posits that the linear sequence of amino acids in a protein dictates its unique three-dimensional structure, and this structure, in turn, determines the protein's specific biological function.

This fundamental link between sequence and function is rooted in the evolutionary principle of homology, the inference of shared ancestry. When a newly discovered sequence exhibits significant similarity to a well-characterized one, it is possible to transfer functional and structural annotations by homology. Sequence alignment serves as the computational formalization of this comparison, constructing a residue-by-residue correspondence to highlight regions of conservation. Because this relationship is probabilistic, rigorous computational methods are required to distinguish statistically significant similarity from random chance.

Further complicating this picture is the modular nature of proteins. Proteins are rarely monolithic entities and are frequently composed of discrete structural and functional units known as domains. These domains act as evolutionary "building blocks" that can be shuffled and combined to create proteins with novel functionalities. This modularity makes the problem of functional inference more tractable, as a robust functional hypothesis can often be built by identifying a protein's constituent domains. The identification of these conserved domains, therefore, is a primary objective of applied bioinformatics and demands algorithms specifically designed to find local regions of high similarity within larger, otherwise dissimilar sequences.

While such local alignment algorithms provide the necessary framework, their accuracy hinges on solving the central challenge of accurately modeling insertions and deletions, also known as gaps. A simple linear gap penalty is computationally efficient but biologically naive. In contrast, a more sophisticated affine gap penalty model offers greater biological realism by distinguishing between the cost of opening and extending a gap. This paper, therefore, presents a rigorous implementation and analysis of the Smith-Waterman local alignment algorithm to demonstrate the superiority of the affine model. This study has three main objectives. First, to implement the Smith-Waterman algorithm with both linear and affine gap penalties. Second, to apply these implementations to a case study involving the identification of the human p53 DNA-binding domain in a Drosophila melanogaster homolog.

Third, to conduct a comparative analysis showing that the affine model produces a more biologically coherent hypothesis of domain conservation.

II. Theoretical Foundations

A. Dynamic Programming

Dynamic Programming is a powerful algorithmic method for solving complex optimization problems by decomposing them into a series of simpler, overlapping subproblems. The solution to the larger problem is then built up from the stored solutions of these subproblems. This approach is characterized by several key features:

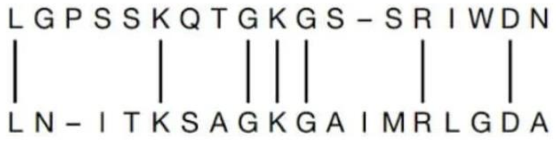
1. Stages and States: A problem suitable for DP can be divided into a sequence of stages. At each stage, a decision is made. Each stage consists of a number of states, which represent the possible conditions or information needed to make a decision at that point. In sequence alignment, the process of filling the alignment matrix column by column (or row by row) can be viewed as progressing through stages, where each cell (i,j) represents a specific state.
2. The Principle of Optimality: This is the cornerstone of dynamic programming. It states that an optimal solution to a multi-stage problem has the property that, whatever the initial state and decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision. In simpler terms, if the total path is optimal, then every sub-path within it must also be optimal. This principle allows us to build an optimal solution iteratively, using the optimal results from previous stages without re-evaluating them.

The implementation of Dynamic Programming typically involves a recursive relationship that connects the optimal solution of a state at stage k to the optimal solutions of states at stage k-1. By solving these recurrences, either in a forward (bottom-up) or backward (top-down) manner, this algorithm guaranteed to find the globally optimal solution to the entire problem.

B. Global Alignment and Local Alignment

Pairwise sequence alignment is dominated by two distinct philosophical and algorithmic approaches that are both built upon dynamic programming. The choice between global and local alignment is not arbitrary, as it represents a fundamental hypothesis about the nature of the relationship between the sequences being compared.

- Global Alignment (The Needleman-Wunsch Algorithm): Global alignment seeks to find the optimal alignment that spans the entire length of both sequences. The foundational algorithm for this task is the Needleman-Wunsch algorithm. The underlying assumption is that the two sequences are homologous across their full length and are of roughly equal size. It is therefore the ideal method for comparing closely related sequences, such as orthologs of the same gene in different species. Its greatest weakness, however, is its application to divergent sequences or those of different lengths, where it will attempt to "force" an alignment across non-homologous regions, often resulting in a biologically meaningless output.



L G P S S K Q T G K G S – S R I W D N
L N – I T K S A G K G A I M R L G D A

Fig. 1. Global Alignment [8]

- Local Alignment (The Smith-Waterman Algorithm): In contrast, local alignment adopts a more focused strategy. Its goal is to identify the single pair of subsequences that yields the highest possible alignment score, making no attempt to align residues outside of this optimal region. The canonical algorithm for this task is the Smith-Waterman algorithm. It is predicated on the idea that even highly dissimilar proteins can share short, conserved regions crucial for function. This makes it the quintessential tool for discovering functional domains and motifs, searching databases, and comparing evolutionarily distant sequences.

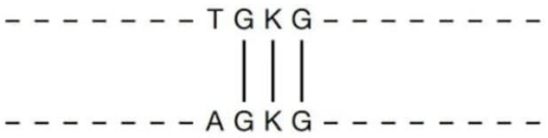


Fig. 2. Local Alignment [8]

C. The Smith-Waterman Algorithm

The Smith-Waterman algorithm provides a rigorous, quantitative method for finding the optimal local alignment between two sequences. It is guaranteed to find the pair of subsequences with the highest possible score, given a specific substitution matrix and gap penalty scheme. Its elegance and power derive from its use of dynamic programming, a computational technique that solves a complex problem by breaking it down into a series of simpler, overlapping subproblems. The algorithm's operation can be understood by examining its three main stages: initialization, matrix filling (recurrence), and traceback.

The dynamic programming matrix can be conceptualized as a directed acyclic graph, where each cell (i, j) is a node. The moves to neighboring cells (diagonal, up, left) represent directed edges, and their weights are determined by the substitution scores and gap penalties. Mismatches and gaps contribute negative weights. Unlike a standard longest-path algorithm, Smith-Waterman introduces a unique rule, the max(..., 0) term. This rule acts as a filter, effectively discarding any alignment path whose cumulative score drops below zero. The algorithm is thus not merely filling a matrix but is conducting an exhaustive search for all possible positively-scoring paths, "islands of profitability" within a "sea of cost." The value zero represents the shoreline, preventing any path from becoming hopelessly negative and allowing the algorithm to abandon a poor alignment and start a new one anywhere in the matrix. This mechanism is the key to its ability to find the most valuable "island," which corresponds to the optimal local alignment.

The Dynamic Programming Matrix

The algorithm's workspace is a two-dimensional matrix, typically denoted as H, with dimensions (m+1) x (n+1), where m and n are the lengths of the two sequences, sequence A and sequence B, respectively. The rows of the matrix correspond to the residues of one sequence, and the columns correspond to the residues of the other. Each cell H(i,j) in the matrix will ultimately store the maximum score of any alignment ending at position i of sequence A and position j of sequence B.

Initialization: The "Free Ride" Start

The initialization step is a defining feature of the Smith-Waterman algorithm and is what distinguishes it fundamentally from global alignment. The entire first row and first column of the matrix H are set to zero.

H(i,0) = 0 for 0 <= i <= m
H(0,j) = 0 for 0 <= j <= n

This initialization is critical for local alignment. It signifies that no penalty is incurred for starting an alignment in the middle of either sequence. In the context of the path-finding analogy, it means that an alignment can begin at any row or column without having to pay an "end-gap" penalty to get there. This "free ride" to any starting point allows the algorithm to identify a high-scoring local alignment that may be embedded deep within two much larger, non-homologous sequences.

The Recurrence Relation: The Heart of the Algorithm

After initialization, the algorithm proceeds to fill the rest of the matrix, typically from the top-left cell H(1,1) to the bottom-right cell H(m,n). The score for each cell H(i,j) is calculated using a recurrence relation that considers the scores of its neighboring cells (to the top, left, and top-left diagonal) and the chosen scoring scheme.

The recurrence relation for the Smith-Waterman algorithm is as follows:

$$H(i,j) = \max \begin{cases} H(i-1,j-1) + s(a_i,b_j) \\ \max_{k\geq 1} \{H(i-k,j) - W_k\} \\ \max_{l\geq 1} \{H(i,j-l) - W_l\} \\ 0 \end{cases}$$

Here, $s(a_i, b_j)$ is the substitution score for aligning residue a_i with b_j (from a matrix like BLOSUM62), and W_k is the penalty for a gap of length k.

The most crucial element of this recurrence is the inclusion of 0 as one of the choices. If the scores derived from all three possible moves (diagonal, up, or left) are negative, it signifies that extending any existing alignment to this point would decrease the overall score. In this case, H(i,j) is set to 0. This action effectively terminates any low-scoring alignment path. This "reset button" is what allows a new, independent local alignment to begin at any point in the matrix, free from the negative-scoring history of its surroundings. While filling the matrix, it is also necessary to keep a traceback matrix that stores pointers indicating which of the four choices led to the score in each cell.

The Traceback Procedure: Reconstructing the Best Local Hit

Once the entire matrix H has been filled, the final step is to identify and reconstruct the optimal local alignment. This process, called the traceback, differs significantly from that of global alignment.

1. Find the Starting Point: The algorithm first scans the entire matrix H to find the cell with the maximum score. This highest score is the score of the optimal local alignment. The coordinates of this cell mark the end of this alignment.

2. Trace the Path: Starting from this maximum-scoring cell, the algorithm traces a path backward through the matrix by following the pointers that were stored during the fill step. If the pointer at H(i,j) points diagonally to H(i-1, j-1), it signifies that a_i was aligned with b_j. If it points up to H(i-1, j), it means a_i was aligned with a gap. If it points left to H(i, j-1), it means b_j was aligned with a gap.

3. Determine the Termination Point: The traceback process continues, following the path of pointers from cell to cell, until it reaches a cell with a score of 0. The point at which the score becomes zero marks the beginning of the optimal local alignment.

The path traced from the highest-scoring cell to the first zero-scoring cell defines the single best local alignment between the two sequences. If multiple local alignments are of interest, the process can be repeated by finding the next highest score in the matrix that is not part of a previously traced path.

D. Scoring Models

The mathematical optimality guaranteed by dynamic programming is meaningless without a scoring scheme that reflects biological reality. The biological relevance of the resulting alignment is critically dependent on the parameters used for scoring. This involves two key components: a substitution matrix to score the alignment of two residues, and a gap penalty model to quantify the cost of insertions or deletions.

Protein Substitution Matrices: BLOSUM62

For protein sequences, a simple match/mismatch score is insufficient because it fails to capture the biochemical and evolutionary realities of amino acid substitutions. Some substitutions between amino acids with similar properties (e.g., size, charge) are often tolerated, while others are highly disruptive. Substitution matrices provide a sophisticated scoring system that reflects these realities, moving from simple identity checks to probabilistic assessments of evolutionary relationships.

The scores within these matrices are founded on a rigorous statistical framework of log-odds ratios. The score for aligning residue i with j reflects the likelihood of that pairing occurring in a truly homologous alignment versus occurring merely by chance. A positive score indicates that a substitution is observed more often than expected, suggesting a functionally tolerated evolutionary change. A negative score indicates a substitution that is observed less often than by chance, suggesting it is deleterious.

The BLOSUM Matrix Family

The BLOSUM (BLOcks SUbstitution Matrix) family, developed by Henikoff and Henikoff in 1992, is derived directly from empirical data and has become the most widely used set of matrices for protein alignment.

- Derivation from BLOCKS: The matrices are derived from the BLOCKS database, a collection of multiple sequence alignments of short, highly conserved, and importantly, ungapped regions (blocks). Using ungapped blocks avoids the confounding effects of gap penalties in the derivation of the substitution scores.
- Clustering to Reduce Redundancy: To mitigate sampling bias from over-represented proteins, sequences within each block that share more than a certain percentage of identity, r%, are clustered and treated as a single sequence. Substitution frequencies are then calculated between these less-redundant clusters.
- Interpreting the BLOSUM Number: The number r in a BLOSUMr matrix refers to this identity threshold. The BLOSUM62 matrix is derived from sequences clustered at ≥62% identity. This makes it ideal for detecting moderately or distantly related proteins. A higher number (e.g., BLOSUM80) is better for closely related sequences, while a lower number (e.g., BLOSUM45) is better for very distant relationships.

BLOSUM62: The De Facto Standard

Among the series, the BLOSUM62 matrix has emerged as the default for many bioinformatics tools (including BLASTp) due to its excellent balance of sensitivity and specificity. Its superiority can be contrasted with the earlier PAM (Point Accepted Mutation) matrices, which were based on an explicit evolutionary model extrapolated from closely related proteins. The BLOSUM approach avoids this extrapolation by deriving scores directly from blocks of varying similarity, better capturing the observed patterns of substitution over long evolutionary timescales. For this study, the standard BLOSUM62 matrix is used.

The Affine Gap Penalty Model

A more realistic approach to scoring gaps is the affine gap penalty model. Its biological rationale is that a single mutational event is more likely to cause a multi-residue indel than multiple separate events. This model assigns a high cost to open a gap (G_{open}) and a lower cost to extend it (G_{extend}). The total penalty for a gap of length d is:

$$G_{total}(d) = G_{open} + (d-1) \times G_{extend}$$

Implementing this model efficiently requires an expansion of the DP state space. Instead of a single matrix, three matrices are used to track the score ending in a Match/Mismatch (M), an insertion (I_x), or a deletion (I_y). This can be conceptualized as a finite state automaton where the alignment can be in one of three states. These matrices are filled using a set of coupled recurrence relations:

$$M(i,j) = s(a_i, b_j) + \max \begin{cases} M(i-1, j-1) \\ I_x(i-1, j-1) \\ I_y(i-1, j-1) \end{cases}$$

$$I_x(i,j) = \max \begin{cases} M(i-1, j) + G_{open} \\ I_x(i-1, j) + G_{extend} \end{cases}$$

$$I_y(i,j) = \max \begin{cases} M(i, j-1) + G_{open} \\ I_y(i, j-1) + G_{extend} \end{cases}$$

The transition from any state at (i-1, j-1) to the match/mismatch state M(i,j) simply adds the substitution score. A transition from the match state M to a gap state (I_x or I_y) incurs the G_{open} penalty. Staying within a gap state incurs the smaller G_{extend} penalty. This structure elegantly prevents the combination of two gaps (e.g., moving from I_x to I_y), which is biologically nonsensical, and correctly models the cost structure.

For local alignment, the final score at cell (i,j) is the maximum of the values from the three matrices, with the addition of the zero term to allow for alignment restarts:

$$H(i,j) = \max\left(M(i,j), I_x(i,j), I_y(i,j), 0\right)$$

The traceback procedure is necessarily more complex, as it must navigate through all three matrices, following the path that led to the maximum score at each step. Despite the increased complexity, this algorithm maintains a time complexity of O(mn), the same as the linear penalty model, while offering significantly greater biological realism.

III. Implementation

A. Program Architecture

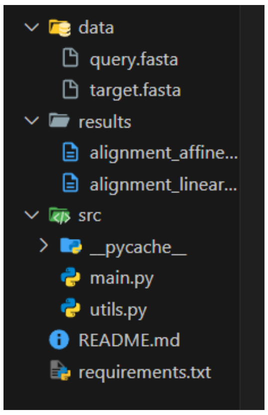
The program was designed with a modular structure in Python to clearly separate data handling, algorithmic computation, and result formatting. The overall workflow proceeds in a linear fashion.

First, the program takes two input files in FASTA format, one containing the query sequence and the other containing the target sequence. A utility module, utils.py, is responsible for parsing these files to extract the raw amino acid sequences.

This module also handles the loading of the standard BLOSUM62 substitution matrix from the Biopython library.

Next, the main execution script, main.py, orchestrates the comparative analysis. It calls the alignment function twice. The first call executes a Smith-Waterman alignment using a simple linear gap penalty model. The second call executes the alignment using the more complex affine gap penalty model.

Finally, upon completion of each alignment, a formatting function is used to generate a human-readable output that includes the final alignment score and the aligned sequences. This output is printed to the console for immediate review and is also saved to a text file in the results/ directory for permanent record.



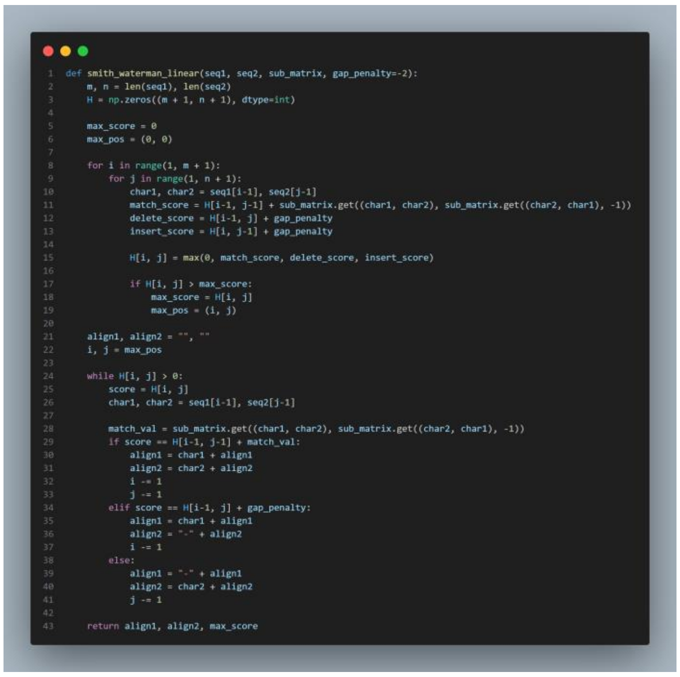
```
data
    query.fasta
    target.fasta
results
    alignment_affine...
    alignment_linear...
src
    __pycache__
    main.py
    utils.py
README.md
requirements.txt
```

Fig. 3. Folder Structure (https://github.com/varel183/Makalah_STIMA_13523008)

B. Algorithmic Implementation

The core of the program consists of two distinct Python functions, each implementing one of the gap penalty models discussed in the theoretical foundations. Both implementations leverage the NumPy library for efficient creation and manipulation of the dynamic programming matrices.

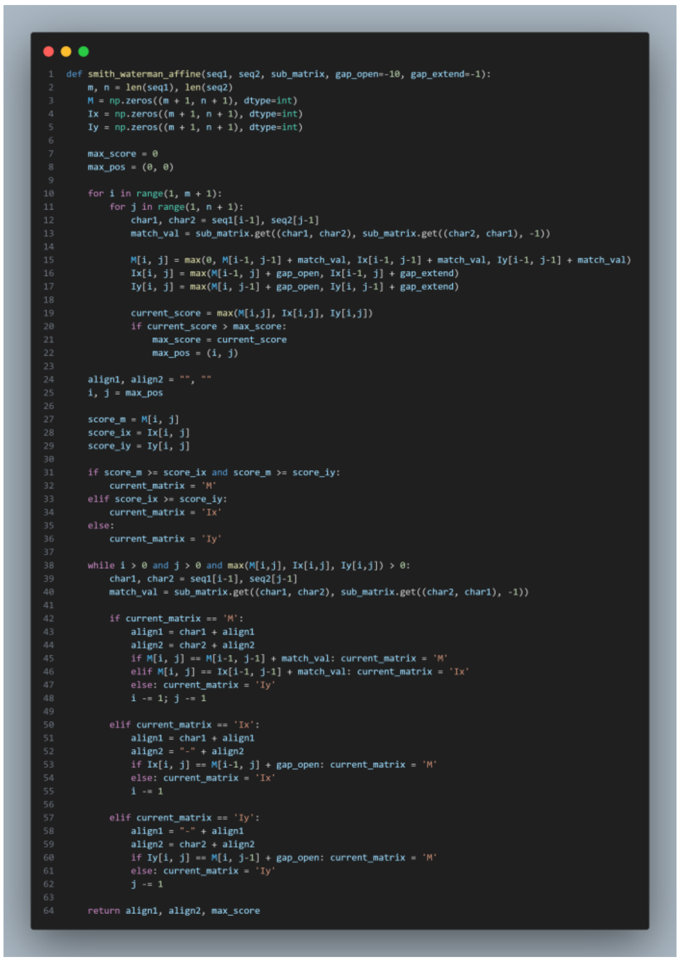
1. Linear Gap Penalty Implementation
   For the baseline linear model, a single two-dimensional matrix, H, of size (m+1) x (n+1) is initialized with zeros. The program then populates this matrix using nested loops that iterate through the sequences. For each cell H(i,j), the score is calculated according to the standard Smith-Waterman recurrence relation, taking the maximum value derived from a diagonal move (match/mismatch), a move from above (deletion), a move from the left (insertion), or zero.



```python
def smith_waterman_linear(seq1, seq2, sub_matrix, gap_penalty=-2):
    m, n = len(seq1), len(seq2)
    H = np.zeros((m + 1, n + 1), dtype=int)

    max_score = 0
    max_pos = (0, 0)

    for i in range(1, m + 1):
        for j in range(1, n + 1):
            char1, char2 = seq1[i-1], seq2[j-1]
            match_score = H[i-1, j-1] + sub_matrix.get((char1, char2), sub_matrix.get((char2, char1), -1))
            delete_score = H[i-1, j] + gap_penalty
            insert_score = H[i, j-1] + gap_penalty

            H[i, j] = max(0, match_score, delete_score, insert_score)

            if H[i, j] > max_score:
                max_score = H[i, j]
                max_pos = (i, j)

    align1, align2 = "", ""
    i, j = max_pos

    while H[i, j] > 0:
        score = H[i, j]
        char1, char2 = seq1[i-1], seq2[j-1]

        match_val = sub_matrix.get((char1, char2), sub_matrix.get((char2, char1), -1))
        if score == H[i-1, j-1] + match_val:
            align1 = char1 + align1
            align2 = char2 + align2
            i -= 1
            j -= 1
        elif score == H[i-1, j] + gap_penalty:
            align1 = char1 + align1
            align2 = "-" + align2
            i -= 1
        else:
            align1 = "-" + align1
            align2 = char2 + align2
            j -= 1

    return align1, align2, max_score
```

Fig. 4. Linear Gap Penalty Implementation (https://github.com/varel183/Makalah_STIMA_13523008/blob/main/src/main.py)

2. Affine Gap Penalty Implementation
   The main implementation for this study addresses the affine gap penalty model. This required an expansion of the dynamic programming state space. Three distinct (m+1) x (n+1) matrices were created using NumPy, corresponding to the three states discussed in the previous chapter.
   - M: Stores the optimal score for an alignment ending in a match or mismatch state.
   - I_x: Stores the optimal score for an alignment ending with a gap in the target sequence (an insertion relative to the query).
   - I_y: Stores the optimal score for an alignment ending with a gap in the query sequence (a deletion relative to the query).



```python
def smith_waterman_affine(seq1, seq2, sub_matrix, gap_open=-10, gap_extend=-1):
    m, n = len(seq1), len(seq2)
    M = np.zeros((m + 1, n + 1), dtype=int)
    Ix = np.zeros((m + 1, n + 1), dtype=int)
    Iy = np.zeros((m + 1, n + 1), dtype=int)

    max_score = 0
    max_pos = (0, 0)

    for i in range(1, m + 1):
        for j in range(1, n + 1):
            char1, char2 = seq1[i-1], seq2[j-1]
            match_val = sub_matrix.get((char1, char2), sub_matrix.get((char2, char1), -1))

            M[i, j] = max(0, M[i-1, j-1] + match_val, Ix[i-1, j-1] + match_val, Iy[i-1, j-1] + match_val)
            Ix[i, j] = max(M[i-1, j] + gap_open, Ix[i-1, j] + gap_extend)
            Iy[i, j] = max(M[i, j-1] + gap_open, Iy[i, j-1] + gap_extend)

            current_score = max(M[i,j], Ix[i,j], Iy[i,j])
            if current_score > max_score:
                max_score = current_score
                max_pos = (i, j)

    align1, align2 = "", ""
    i, j = max_pos

    score_m = M[i, j]
    score_ix = Ix[i, j]
    score_iy = Iy[i, j]

    if score_m >= score_ix and score_m >= score_iy:
        current_matrix = 'M'
    elif score_ix >= score_iy:
        current_matrix = 'Ix'
    else:
        current_matrix = 'Iy'

    while i > 0 and j > 0 and max(M[i,j], Ix[i,j], Iy[i,j]) > 0:
        char1, char2 = seq1[i-1], seq2[j-1]
        match_val = sub_matrix.get((char1, char2), sub_matrix.get((char2, char1), -1))

        if current_matrix == 'M':
            align1 = char1 + align1
            align2 = char2 + align2
            if M[i, j] == M[i-1, j-1] + match_val: current_matrix = 'M'
            elif M[i, j] == Ix[i-1, j-1] + match_val: current_matrix = 'Ix'
            else: current_matrix = 'Iy'
            i -= 1; j -= 1

        elif current_matrix == 'Ix':
            align1 = char1 + align1
            align2 = "-" + align2
            if Ix[i, j] == M[i-1, j] + gap_open: current_matrix = 'M'
            else: current_matrix = 'Ix'
            i -= 1

        elif current_matrix == 'Iy':
            align1 = "-" + align1
            align2 = char2 + align2
            if Iy[i, j] == M[i, j-1] + gap_open: current_matrix = 'M'
            else: current_matrix = 'Iy'
            j -= 1

    return align1, align2, max_score
```

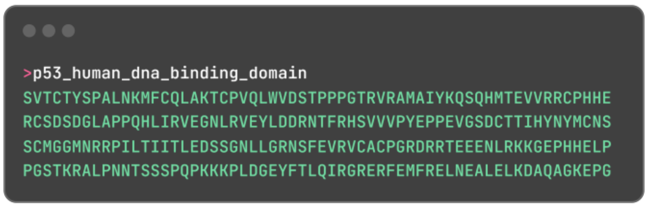
Fig. 5. Affine Gap Penalty Implementation (https://github.com/varel183/Makalah_STIMA_13523008/blob/main/src/main.py)

The traceback procedure for the affine model is necessarily more complex. It begins at the max_pos found during the fill stage. At each step, it determines which of the three matrices (M, I_x, or I_y) contributed to the score, and then checks the recurrence relation for that specific matrix to decide the preceding move. This process is repeated until a cell with a score of zero is reached.

C. Experiments

The algorithms were applied to a specific biological case study to compare their performance.

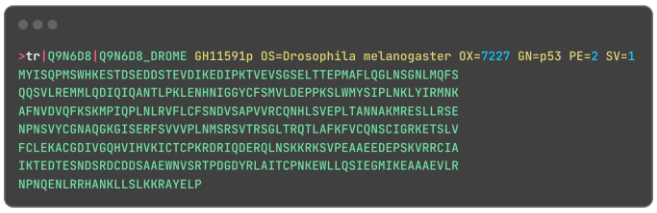
- Query Sequence: The query was the DNA-binding domain of the human p53 tumor suppressor protein (residues 102-292). This specific subsequence was extracted from the canonical human p53 protein, which corresponds to UniProt accession number P04637.



```
>p53_human_dna_binding_domain
SVTCTYSPALNKMFCQLAKTCPVQLWVDSTPPPGTRVRAMAIYKQSQHMTEVVRRCPHHE
RCSDSDGLAPPQHLIRVEGNLRVEYLDDRNTFRHSVVVPYEPPEVGSDCTTIHYNYMCNS
SCMGGMNRRPILTIITLEDSSGNLLGRNSFEVRVCACPGRDRRTEEENLRKKGEPHHELP
PGSTKRALPNNTSSSPQPKKKPLDGEYFTLQIRGRERFEMFRELNEALELKDAQAGKEPG
```

Fig. 6. Query Sequence (https://www.uniprot.org/uniprotkb/P04637/entry)

- Target Sequence: The target was the full-length p53 protein homolog from the fruit fly, Drosophila melanogaster. This corresponds to UniProt accession number Q9N6D8.



```
>tr|Q9N6D8|Q9N6D8_DROME GH11591p OS=Drosophila melanogaster OX=7227 GN=p53 PE=2 SV=1
MYISQPMSWHKESTDSEDDSTEVDIKEDIPKTVEVSGSELTTEPMAFLQGLNSGNLMQFS
QQSVLREMMLQDIQIQANTLPKLENHNIGGYCFSMVLDEPPKSLWMYSIPLNKLYIRMNK
AFNVDVQFKSKMPIQPLNLRVFLCFSNDVSAPVVRCQNHLSVEPLTANNAKMRESLLRSE
NPNSVYCGNAQGKGISERFSVVVPLNMSRSVTRSGLTRQTLAFKFVCQNSCIGRKETSLV
FCLEKACGDIVGQHVIHVKICTCPKRDRIQDERQLNSKKRKSVPEAAEEDEPSKVRRCIA
IKTEDTESNDSRDCDDSAAEWNVSRTPDGDYRLAITCPNKEWLLQSIEGMIKEAAAEVLR
NPNQENLRRHANKLLSLKKRAYELP
```

Fig. 7. Target Sequence (https://www.uniprot.org/uniprotkb/Q9N6D8/entry)

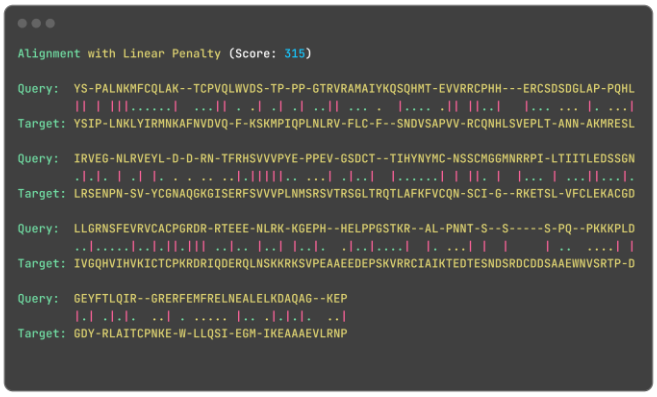
The following scoring parameters were used for the two comparative experiments:

- Substitution Matrix: The standard BLOSUM62 matrix was used for both alignment models to score amino acid substitutions.
- Linear Gap Penalty: For the baseline model, a linear gap penalty of -2 was applied.
- Affine Gap Penalties: For the main experimental model, a gap opening penalty of -10 and a gap extension penalty of -1 were used.

IV. Result and Analysis

A. Alignment with a Linear Gap Penalty

The initial experiment was conducted using a standard linear gap penalty, where the penalty for any gap, regardless of its length, was set to w = -2. The algorithm processed the query sequence (human p53 DNA-binding domain) and the target sequence (Drosophila melanogaster p53) and produced a maximal alignment score of 315. The optimal local alignment generated by this model is presented in Figure 8.



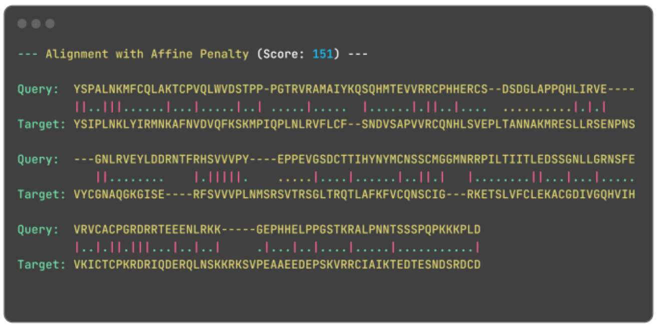
```
Alignment with Linear Penalty (Score: 315)

Query:  YS-PALNKMFCQLAK--TCPVQLWVDS-TP-PP-GTRVRAMAIYKQSQHMT-EVVRRCPHH---ERCSDSDGLAP-PQHL
        || | |||......|  ...|| . .| .| .| ..|| ... .  |.... .|| ||..|   |... ... |. ...|
Target: YSIP-LNKLYIRMNKAFNVDVQ-F-KSKMPIQPLNLRV-FLC-F--SNDVSAPVV-RCQNHLSVEPLT-ANN-AKMRESL

Query:  IRVEG-NLRVEYL-D-D-RN-TFRHSVVVPYE-PPEV-GSDCT--TIHYNYMC-NSSCMGGMNRRPI-LTIITLEDSSGN
        .|.|. | .| |. . . .. ..|.||||.. ...| .|..|  |......| | ||. |  |... | ...||...|.
Target: LRSENPN-SV-YCGNAQGKGISERFSVVVPLNMSRSVTRSGLTRQTLAFKFVCQN-SCI-G--RKETSL-VFCLEKACGD

Query:  LLGRNSFEVRVCACPGRDR-RTEEE-NLRK-KGEPH--HELPPGSTKR--AL-PNNT-S--S-----S-PQ--PKKKPLD
        ..|.....|..|.||.||| ..|.. |..| |..|.  .|..|....|  |. ...| |  |     | ..  ....| |
Target: IVGQHVIHVKICTCPKRDRIQDERQLNSKKRKSVPEAAEEDEPSKVRRCIAIKTEDTESNDSRDCDDSAAEWNVSRTP-D

Query:  GEYFTLQIR--GRERFEMFRELNEALELKDAQAG--KEP
        |.| .|.|.  ..| . ..... |.. .|.|.|.  ..|
Target: GDY-RLAITCPNKE-W-LLQSI-EGM-IKEAAAEVLRNP
```

Fig. 8. Local Alignment using a Linear Gap Penalty (https://github.com/varel183/Makalah_STIMA_13523008/blob/main/results/alignment_linear.txt)

A visual inspection of the alignment in Figure 8 immediately reveals a key characteristic, the alignment is fragmented by numerous, small, scattered gaps. This "peppering" of gaps is evident throughout the alignment, with examples such as D-D-RN-TFR and VDS-TP-PP-GTRV. This pattern is a direct consequence of the linear penalty model, where the cost to open a new gap is identical to the cost of extending one. The algorithm, therefore, has no incentive to group gaps and will frequently insert single gaps to achieve minor increases in the substitution score. From a biological standpoint, this scenario is less plausible as it implies a large number of independent, single-residue indel events, which are evolutionarily less frequent than single, larger indel events that might occur in a single mutational step.

B. Alignment with an Affine Gap Penalty

The second experiment utilized the affine gap penalty model with parameters set to $G_{open} = -10$ and $G_{extend} = -1$. This model produced an optimal local alignment score of 151. The resulting alignment is shown in Figure 9.



```
--- Alignment with Affine Penalty (Score: 151) ---

Query:  YSPALNKMFCQLAKTCPVQLWVDSTPP-PGTRVRAMAIYKQSQHMTEVVRRCPHHERCS--DSDGLAPPQHLIRVE----
        ||..|||......|...|.....|..| .....|.....  |......|.||..|....  ..........|.|.|
Target: YSIPLNKLYIRMNKAFNVDVQFKSKMPIQPLNLRVFLCF--SNDVSAPVVRCQNHLSVEPLTANNAKMRESLLRSENPNS

Query:  ---GNLRVEYLDDRNTFRHSVVVPY----EPPEVGSDCTTIHYNYMCNSSCMGGMNRRPILTIITLEDSSGNLLGRNSFE
           ||........    |.|||||.    .....|....|......|..||.|   |........||...|...|.....
Target: VYCGNAQGKGISE----RFSVVVPLNMSRSVTRSGLTRQTLAFKFVCQNSCIG---RKETSLVFCLEKACGDIVGQHVIH

Query:  VRVCACPGRDRRTEEENLRKK-----GEPHHELPPGSTKRALPNNTSSSPQPKKKPLD
        |..|.||.|||...|..|..|     .|...|..|....|.....|...........|
Target: VKICTCPKRDRIQDERQLNSKKRKSVPEAAEEDEPSKVRRCIAIKTEDTESNDSRDCD
```

Fig. 9. Local Alignment using an Affine Gap Penalty (https://github.com/varel183/Makalah_STIMA_13523008/blob/main/results/alignment_affine.txt)

The structure of this alignment is markedly different from the one produced by the linear model. The most notable feature is the consolidation of gaps into larger, more cohesive blocks. Clear examples include IRVE----GNLR, VPY----EPPEV, and RKK-----GEPH. This is the intended outcome of the affine model. The high cost of opening a gap (-10) discourages the creation of new gaps, while the much lower extension cost (-1) incentivizes the lengthening of existing ones. This structure better represents a more probable evolutionary scenario, such as a single insertion or deletion event affecting a contiguous block of amino acids. Such events often correspond to changes in the structurally flexible loop regions of a protein, which connect its core secondary structure elements.

C. Comparative Analysis

Comparing the outputs from the two models (Figure 8 and 9) highlights the profound impact of the scoring scheme on the resulting biological hypothesis. Although the linear model produced a numerically higher score (315 vs. 151), this is a mathematical artifact of its less restrictive nature. The affine gap model, despite its lower score, produces an alignment that is structurally and biologically far more coherent.

The alignment generated by the affine model presents a more robust and parsimonious evolutionary hypothesis. The consolidation of gaps into coherent blocks suggests that the core structural elements of the p53 DNA-binding domain are preserved between human and Drosophila, with indel events largely confined to the loop regions that connect them. This is the expected pattern for a conserved functional domain.

Furthermore, an examination of the conserved columns (indicated by |) in the affine alignment reveals the preservation of key functional residues. For instance, the alignment correctly pairs several Cysteine (C) and Arginine (R) residues, which are well-documented in the literature as being critical for the structural integrity (via zinc coordination) and DNA-contact functions of the p53 DNA-binding domain. The linear model's fragmented alignment obscures some of these key correspondences.

In conclusion, the affine model's ability to produce an alignment that correctly highlights the conservation of non-negotiable functional sites, while confining gaps to likely non-critical regions, provides strong evidence for its superiority in this context. It generates a more trustworthy computational hypothesis, suggesting that the Drosophila p53 protein indeed contains a domain that is not only similar in sequence but also likely conserved in both structure and function to its human counterpart.

V. CONCLUSION

This paper presented a rigorous implementation and a comparative analysis of the Smith-Waterman local alignment algorithm under two distinct gap penalty frameworks. The study systematically compared the outputs of a simple linear gap penalty model against those of a more complex, biologically-motivated affine gap penalty model. Through a practical case study focused on identifying the conserved DNA-binding domain of the human p53 protein within its Drosophila melanogaster homolog, this work aimed to demonstrate how the underlying mathematical model directly influences the quality and biological interpretability of sequence alignments. The implementation was successfully developed in Python, leveraging the NumPy library for efficient matrix computation, and was used to generate alignment data for both models.

The central finding of this study is that the affine gap penalty model, despite often yielding a lower numerical score, produces alignments that are demonstrably superior in biological relevance. While the linear model generated fragmented alignments peppered with small, scattered gaps, the affine model successfully consolidated these into larger, contiguous blocks. This outcome aligns more closely with the current understanding of molecular evolution, where single, large-scale insertion or deletion events affecting structural loops are considered more probable than numerous, independent single-residue mutations. Consequently, the affine gap penalty model generates a more robust and structurally coherent alignment, providing a stronger and more trustworthy computational hypothesis for the presence and boundaries of a conserved functional domain.

While this study successfully demonstrated the superiority of the affine model in a specific and important case, several avenues for future research remain open. First, the performance of the Python implementation, while suitable for this analysis, could be significantly optimized for large-scale database searches. This could be achieved through reimplementation in a lower-level compiled language like C++ or by exploring parallelization techniques. Second, the conclusions could be further validated by applying this comparative analysis to a broader range of protein families with different evolutionary rates and structural characteristics. Finally, future work could explore even more sophisticated scoring schemes, such as context-specific or convex gap penalty models, to investigate whether further gains in biological accuracy can be achieved, continuing the pursuit of computational models that more perfectly mirror evolutionary reality.

VIDEO LINK AT YOUTUBE

https://youtu.be/4oS7rDhzv0k

GITHUB

https://github.com/varel183/Makalah_STIMA_13523008

ACKNOWLEDGMENT

The author wishes to express sincere gratitude to the lecturing team of the IF2211 Algorithm Strategies course for the invaluable guidance and knowledge provided throughout the semester. Special thanks are extended to Dr. Rinaldi as the course coordinator, as well as to Dr. Nur Ulfa Maulidevi and Monterico Adrian, M.T., for their lectures and support during the second semester of the 2024/2025 academic year. Their instruction formed the essential foundation upon which this work was built.

This research would not have been possible without the publicly available data from the UniProt Consortium and the National Center for Biotechnology Information (NCBI). Furthermore, the implementation of the algorithms relied on the powerful open-source libraries NumPy and Biopython, and the author acknowledges the contributions of their respective development communities. Finally, thanks are also due to friends and colleagues for the insightful discussions and collaborative learning environment.

REFERENCES

[1] T.F. Smith and M.S. Waterman, “Identification of common molecular subsequences,” J. Mol. Biol., vol. 147, no. 1, pp. 195-197, Mar. 1981.

[2] O. Gotoh, “An improved algorithm for matching biological sequences,” J. Mol. Biol., vol. 162, no. 3, pp. 705-708, Dec. 1982.

[3] S.B. Needleman and C.D. Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins,” J. Mol. Biol., vol. 48, no. 3, pp. 443-453, Mar. 1970.

[4] S. Henikoff and J.G. Henikoff, “Amino acid substitution matrices from protein blocks,” Proc. Natl. Acad. Sci. U.S.A., vol. 89, no. 22, pp. 10915-10919, Nov. 1992.

[5] S.F. Altschul, T.L. Madden, A.A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman, “Gapped BLAST and PSI-BLAST: a new generation of protein database search programs,” Nucleic Acids Res., vol. 25, no. 17, pp. 3389-3402, Sep. 1997.

[6] The UniProt Consortium, "UniProt: the universal protein knowledgebase in 2023," Nucleic Acids Res., vol. 51, no. D1, pp. D523-D531, Jan. 2023.

[7] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison, Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids. Cambridge: Cambridge University Press, 1998, pp. 12-45.

[8] Mount, D. W. (2001) Bioinformatics: sequence and genome analysis. Cold Spring Harbor Laboratory Press.

[9] R. Munir, "Program Dinamis (Dynamic Programming) Bagian 1," Bahan Kuliah IF2211 Strategi Algoritma, Program Studi Teknik Informatika, STEI-ITB, 2025. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/25-Program-Dinamis-(2025)-Bagian1.pdf. [Accessed: Jun. 23, 2025].

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 24 Juni 2025

[illegible]

Varel Tiara - 13523008