Ejercicio 3: Alineación de Secuencias de Nucleótidos

Algoritmo de Needleman-Wunsch

Fabricio Denardi

Diciembre 2025

1. Parte 1: Conceptos Teóricos

1.1. Secuencias de Nucleótidos y su Importancia

Una secuencia de nucleótidos es una cadena lineal de nucleótidos que constituyen el ADN o ARN de un organismo. En el caso del ADN, está compuesta por cuatro bases nitrogenadas: adenina (A), timina (T), guanina (G) y citosina (C). En el ARN, la timina es reemplazada por uracilo (U). Estas secuencias contienen la informacion genética que codifica para proteinas y regula procesos celulares fundamentales.

La comparacion de secuencias de nucleótidos es crucial en biologia molecular por varias razones:

- Relaciones evolutivas: secuencias similares sugieren ancestros comunes y permiten construir arboles filogenéticos.
- Identificación de genes: permite encontrar genes homólogos en diferentes organismos y predecir sus funciones.
- Diagnóstico médico: ayuda a identificar mutaciones asociadas a enfermedades geneticas.
- Biotecnología: facilita el diseño de primers para PCR, desarrollo de vacunas y terapias génicas.

1.2. Alineación de Secuencias

La alineación de secuencias es el proceso de disponer dos o más secuencias biológicas (ADN, ARN o proteínas) de manera que se identifiquen regiones de similitud que puedan indicar relaciones funcionales, estructurales o evolutivas entre ellas.

1.2.1. Tipos de Alineación

Alineación Global: compara las secuencias completas de principio a fin. Es útil cuando las secuencias tienen longitudes similares y se espera que sean homólogas en toda su extensión. El algoritmo de Needleman-Wunsch implementa este tipo de alineación.

Alineación Local: identifica regiones de similitud dentro de secuencias más largas, sin necesidad de alinear las secuencias completas. Es útil cuando solo dominios o regiones especificas son similares. El algoritmo de Smith-Waterman implementa este enfoque.

1.2.2. Elementos de un Alineamiento

- Coincidencias (matches): posiciones donde los nucleótidos de ambas secuencias son identicos. Indican conservación evolutiva.
- Desajustes (mismatches): posiciones donde los nucleótidos difieren. Pueden representar mutaciones puntuales (sustituciones) ocurridas durante la evolucion.
- Huecos (gaps): representan inserciones o eliminaciones (indels) en una secuencia respecto a la otra. Se introducen para optimizar el alineamiento y reflejan eventos evolutivos de inserción o eliminación de nucleótidos.

1.3. Modelo de Puntuación

El modelo de puntuación define el çostorelativo de diferentes eventos evolutivos y determina la calidad del alineamiento.

1.3.1. Reglas de Puntuación Básicas

- Match: se asigna una puntuación positiva cuando dos nucleótidos coinciden. En el esquema simple: $+1$
- Mismatch: se asigna una penalización cuando dos nucleótidos no coinciden. En el esquema simple: -1

En modelos más sofisticados, se utilizan matrices de sustitucion (como BLOSUM o PAM para proteínas) que consideran la probabilidad evolutiva de cada tipo de mutación específica.

1.3.2. Penalización por Huecos

La penalizacion por gaps es crucial para evitar alineamientos excesivos con multiples inserciones/eliminaciones:

- Penalización por apertura de gap: costo inicial de introducir un nuevo gap. En el esquema simple: -2
- Penalización por extensión de gap: costo adicional por cada posición que se extiende un gap existente. Muchos algoritmos usan penalizaciones afines: penalty $= d + e \times k$, donde d es el costo de apertura, e el costo de extensión, y k la longitud del gap.

En nuestro esquema simplificado, cada gap tiene un costo fijo de -2, independientemente de su longitud.

1.4. Algoritmo de Needleman-Wunsch

El algoritmo de Needleman-Wunsch (1970) es un método de programación dinámica que garantiza encontrar el alineamiento global óptimo entre dos secuencias. Es un algoritmo fundamental en bioinformatica.

1.4.1. Construcción de la Matriz de Programación Dinámica

El algoritmo construye una matriz M de dimensiones $(n+1) \times (m+1)$, donde n y m son las longitudes de las dos secuencias.

Inicialización:

- La celda $M[0,0] = 0$ (secuencias vacias)
- Primera fila: $M[0,j] = j \times \text{gap\_penalty}$ para $j = 1, \ldots, m$
- Primera columna: $M[i,0] = i \times \text{gap\_penalty}$ para $i = 1, \ldots, n$

Estas condiciones de frontera representan el costo de alinear una secuencia con la secuencia vacía (solo gaps).

1.4.2. Ecuación de Recurrencia

Para cada celda $M[i,j]$, se calcula el score óptimo considerando tres posibilidades:

$$M[i,j] = \text{máx} \begin{cases} M[i-1,j-1] + s(x_i, y_j) & \text{(diagonal: match/mismatch)} \\ M[i-1,j] + \text{gap} & \text{(arriba: gap en seq2)} \\ M[i,j-1] + \text{gap} & \text{(izquierda: gap en seq1)} \end{cases} \tag{1}$$

donde $s(x_i, y_j)$ es la puntuación de emparejar los caracteres x_i y y_j:

$$s(x_i, y_j) = \begin{cases} \text{match} & \text{si } x_i = y_j \\ \text{mismatch} & \text{si } x_i \neq y_j \end{cases} \tag{2}$$

Esta ecuación implementa el principio de optimalidad de Bellman: la solución óptima de un problema se puede construir a partir de soluciones óptimas de sus subproblemas.

1.4.3. Proceso de Traceback

Una vez construida la matriz completa, el score óptimo se encuentra en $M[n,m]$. Para recuperar el alineamiento que produjo ese score:

1. Inicio: comenzar en la celda $M[n,m]$ (esquina inferior derecha)
2. Recorrido: en cada paso, determinar de qué celda provino el valor actual:
   - Si $M[i,j]$ vino de $M[i-1,j-1]$: alinear x_i con y_j (movimiento diagonal)
   - Si $M[i,j]$ vino de $M[i-1,j]$: alinear x_i con gap (movimiento vertical)
   - Si $M[i,j]$ vino de $M[i,j-1]$: alinear gap con y_j (movimiento horizontal)
3. Terminación: continuar hasta llegar a $M[0,0]$
4. Reversión: como el alineamiento se construyó de atrás hacia adelante, invertirlo para obtener el orden correcto

1.4.4. Complejidad Computacional

El algoritmo tiene:

- Complejidad temporal: $O(n \times m)$ para construir la matriz
- Complejidad espacial: $O(n \times m)$ para almacenar la matriz

Para secuencias muy largas, existen optimizaciones que reducen el espacio a $O(\text{mín}(n, m))$.

2. Parte 2: Implementación del Algoritmo

2.1. repositorio

El repositorio del código puede encontrarse en: `https://github.com/denardifabricio/MIA_01c_CAED/tree/main/Ejercicio3`

2.2. Descripción General del Código

Los invito a revisar el código completo en el archivo `needleman_wunsch.py` en el repositorio de código, sin embargo, a modo de resumen: Se implementó el algoritmo de Needleman-Wunsch en Python para realizar alineamientos globales de secuencias de nucleótidos. El programa está estructurado de manera modular con las siguientes funciones principales:

2.2.1. Función `needleman_wunsch()`

Esta es la función central que implementa el algoritmo completo:

1. Inicialización: crea una matriz de dimensiones $(n + 1) \times (m + 1)$ e inicializa las condiciones de frontera con penalizaciones acumuladas de gaps.
2. Llenado de la matriz: implementa la ecuación de recurrencia iterando sobre cada celda y calculando el score óptimo considerando los tres movimientos posibles (diagonal, arriba, izquierda).
3. Traceback: reconstruye el alineamiento óptimo recorriendo la matriz desde la esquina inferior derecha hasta el origen, determinando en cada paso qué movimiento produjo el valor actual.
4. Retorno: devuelve la matriz completa, ambas secuencias alineadas y el puntaje final.

Código de inicialización de la matriz:

```
# Crear matriz de dimensiones (n+1) x (m+1)
n_rows = len(seq1)
n_cols = len(seq2)
matriz = [[0 for _ in range(n_cols + 1)]
          for _ in range(n_rows + 1)]

# Inicializar primera fila y columna con penalizaciones
for i in range(n_rows + 1):
```

```
    matriz[i][0] = i * gap
for j in range(n_cols + 1):
    matriz[0][j] = j * gap
```

Listing 1: Inicialización de la matriz y condiciones de frontera

Código de llenado de la matriz con programación dinámica:

```
for i in range(1, n_rows + 1):
    for j in range(1, n_cols + 1):
        # Calcular score diagonal (match o mismatch)
        if seq1[i-1] == seq2[j-1]:
            diagonal_score = matriz[i-1][j-1] + match
        else:
            diagonal_score = matriz[i-1][j-1] + mismatch

        # Calcular scores con gaps
        up_score = matriz[i-1][j] + gap      # Gap en seq2
        left_score = matriz[i][j-1] + gap    # Gap en seq1

        # Tomar el maximo (decision optima)
        matriz[i][j] = max(diagonal_score, up_score,
                           left_score)
```

Listing 2: Ecuación de recurrencia - Llenado de la matriz

Código del proceso de traceback:

```
alignment1 = []
alignment2 = []
i = n_rows
j = n_cols

while i > 0 or j > 0:
    if j == 0:
        # Solo gaps en seq2
        alignment1.append(seq1[i-1])
        alignment2.append('-')
        i -= 1
    elif i == 0:
        # Solo gaps en seq1
        alignment1.append('-')
        alignment2.append(seq2[j-1])
        j -= 1
    else:
        # Determinar de donde vino el valor actual
        if seq1[i-1] == seq2[j-1]:
            diagonal_score = matriz[i-1][j-1] + match
        else:
            diagonal_score = matriz[i-1][j-1] + mismatch

        up_score = matriz[i-1][j] + gap
        left_score = matriz[i][j-1] + gap

```

```
        # Elegir el camino que produjo el valor
        if matriz[i][j] == diagonal_score:
            alignment1.append(seq1[i-1])
            alignment2.append(seq2[j-1])
            i -= 1
            j -= 1
        elif matriz[i][j] == up_score:
            alignment1.append(seq1[i-1])
            alignment2.append('-')
            i -= 1
        else:
            alignment1.append('-')
            alignment2.append(seq2[j-1])
            j -= 1

# Invertir los alineamientos
alignment1 = ''.join(reversed(alignment1))
alignment2 = ''.join(reversed(alignment2))
```

Listing 3: Traceback para recuperar el alineamiento óptimo

2.2.2. Funciones de Visualización

- `print_matrix()`: imprime la matriz de puntuación de forma tabular, mostrando los encabezados de ambas secuencias para facilitar la interpretacion.
- `print_alignment()`: visualiza el alineamiento resultante con simbolos intuitivos:
  - `|` para matches (coincidencias)
  - `*` para mismatches (desajustes)
  - espacio para gaps (huecos)
- `analyze_alignment()`: calcula y muestra estadisticas detalladas del alineamiento incluyendo número de matches, mismatches, gaps, y porcentaje de identidad.

Código para análisis de estadísticas:

```
num_matches = 0
num_mismatches = 0
num_gaps = 0

for i in range(len(alignment1)):
    if alignment1[i] == '-' or alignment2[i] == '-':
        num_gaps += 1               # Gap
    elif alignment1[i] == alignment2[i]:
        num_matches += 1            # Match
    else:
        num_mismatches += 1         # Mismatch

longitud_total = len(alignment1)
porcentaje_identidad = (num_matches / longitud_total) * 100

```

```
print(f"Longitud del alineamiento: {longitud_total}")
print(f"Coincidencias (matches):   {num_matches}")
print(f"Desajustes (mismatches):   {num_mismatches}")
print(f"Huecos (gaps):             {num_gaps}")
print(f"Identidad:                 {porcentaje_identidad:.2f}%")
```

Listing 4: Análisis estadístico del alineamiento

2.2.3. Esquema de Puntuación Implementado

El programa utiliza el siguiente modelo de puntuación simple:

- Match: +1 (recompensa por nucleótidos idénticos)
- Mismatch: −1 (penalización por nucleótidos diferentes)
- Gap: −2 (penalización por inserción/eliminación)

Este esquema favorece las coincidencias y penaliza más fuertemente los gaps que los mismatches, lo cual es apropiado para secuencias relacionadas evolutivamente donde las inserciones/eliminacions son eventos menos frecuentes que las sustituciones puntuales.

Código de definición de parámetros:

```
def needleman_wunsch(seq1, seq2, match=1, mismatch=-1, gap=-2):
    """
    Implementa el algoritmo de Needleman-Wunsch

    Parametros:
    - match: +1 (recompensa por coincidencia)
    - mismatch: -1 (penalizacion por desajuste)
    - gap: -2 (penalizacion por hueco)
    """
    # ... implementacion del algoritmo ...
```

Listing 5: Parámetros del modelo de puntuación

Ejemplo de uso del programa:

```
# Definir parejas de secuencias a analizar
parejas_secuencias = [
    ("GATTACA", "GCATGCU"),     # Caso clasico
    ("ACGT", "ACCT"),           # Secuencias cortas
    ("ATGCT", "AGCT"),          # Requiere gaps
]

# Parametros de puntuacion
MATCH = 1
MISMATCH = -1
GAP = -2

# Procesar cada pareja
```

```
for seq1, seq2 in parejas_secuencias:
    matriz, alineamiento1, alineamiento2, puntaje = \
        needleman_wunsch(seq1, seq2, MATCH, MISMATCH, GAP)

    # Mostrar resultados
    print_matrix(matriz, seq1, seq2)
    print_alignment(alineamiento1, alineamiento2, seq1, seq2)
    analyze_alignment(alineamiento1, alineamiento2,
                      MATCH, MISMATCH, GAP)
```

Listing 6: Ejemplo de ejecución con parejas de secuencias

2.3. Casos de Prueba

El programa procesa 10 parejas de secuencias que cubren diversos escenarios:

1. (”GATTACA”, ”GCATGCU”): caso clásico de referencia con multiples mismatches
2. (.ACGT”, .ACCT”): secuencias cortas con un unico mismatch
3. (.ATGCT”, .AGCT”): requiere introducir un gap óptimo
4. Casos adicionales con secuencias de diferente longitud, alta similitud, múltiples gaps, etc.

Para cada pareja, el programa muestra:

- La matriz de puntuación completa
- El alineamiento global óptimo
- Estadísticas detalladas (matches, mismatches, gaps, identidad)
- El puntaje final del alineamiento

2.4. Características Técnicas

- Lenguaje: Python 3
- Complejidad temporal: $O(n \times m)$ donde n y m son las longitudes de las secuencias
- Complejidad espacial: $O(n \times m)$ para almacenar la matriz completa
- Documentación: el código incluye comentarios extensos explicando la teoria detrás de cada paso
- Modularidad: funciones separadas para diferentes aspectos del analisis, facilitando mantenimiento y reutilización

2.5. Ejemplo de Resultado

Para la pareja (`"GATTACA", "GCATGCU"`), el algoritmo produce:

```
Alineamiento óptimo:
Seq1: GATTACA
      |**|*|*
Seq2: GCATGCU

Estadísticas:
- Longitud: 7
- Matches: 3 (+3 puntos)
- Mismatches: 4 (-4 puntos)
- Gaps: 0 (+0 puntos)
- Identidad: 42.86%
- Puntaje final: -1
```

Este resultado muestra que aunque las secuencias comparten algunos nucleótidos conservados, presentan diferencias significativas que resultan en un score negativo, indicando baja similitud global.