Università degli Studi di Salerno

DIPARTIMENTO DI INFORMATICA

CORSO DI LAUREA MAGISTRALE IN CYBERSECURITY

Relazione sugli Algoritmi Classici e Moderni di Allineamento delle Sequenze

Professori:

Rosalba Zizza

Rocco Zaccagnino

Clelia De Felice

Studenti:

Marco Fusco

Marco Palmisciano

Vittorio Ciancio

Anno Accademico 2024 - 2025

Indice

Capitolo 1

Introduzione

L'**allineamento di sequenze** (*Sequence Alignment, SA*) è uno strumento cruciale in bioinformatica e biologia computazionale per confrontare sequenze biologiche (DNA, RNA e proteine) e analizzare similarità strutturali e funzionali. Questo processo è alla base di molte applicazioni, tra cui:

- Diagnosi precoce di malattie genetiche;
- Sviluppo di farmaci mirati;
- Studio della genetica evolutiva e costruzione di alberi filogenetici;
- Identificazione forense in ambito criminale.

1.1 Bioinformatica e Biologia Computazionale

La bioinformatica, un campo interdisciplinare che integra biologia e scienze computazionali, crea algoritmi, database e strumenti per analizzare grandi quantità di dati biologici. In questo contesto, l'allineamento di sequenze permette di:

- Identificare similarità evolutive;
- Quantificare distanze filogenetiche;
- Rilevare polimorfismi e mutazioni genetiche.

1.2 Classificazione delle Sequenze

Le sequenze biologiche sono rappresentate come stringhe di simboli:

- **Acidi nucleici:** il DNA è composto da Adenina (A), Guanina (G), Citosina (C) e Timina (T), mentre l'RNA sostituisce la Timina con Uracile (U).

- **Proteine:** composte da 20 amminoacidi, rappresentati da lettere dell'alfabeto (A, C, D, E, . . . , Y).

Le differenze tra sequenze derivano principalmente da:

1. ***Sostituzioni:*** cambio di una base con un'altra.
2. ***Inserzioni/Delezioni:*** aggiunta o rimozione di una o più basi.

Questi fattori complicano il confronto e richiedono algoritmi avanzati per gestire gap introdotti durante l'allineamento.

1.3 Tipologie di Allineamento

Gli allineamenti possono essere classificati in base alla loro estensione:

- **Globale:** confronta l'intera lunghezza delle sequenze.

Tabella 1.1: Esempio di Allineamento Globale

Descrizione	**Sequenza**
Sequenza 1 di partenza	G A T C G A T A G C
Sequenza 2 di partenza	G A T G A T A G C
Allineamento finale	G A T C G A T A G C G A T - G A T A G C

- **Locale:** confronta regioni altamente simili.

Tabella 1.2: Esempio di Allineamento Locale

Descrizione	**Sequenza**
Sequenza 1 di partenza	G A T C G A T A G C
Sequenza 2 di partenza	G A T G A T A G C
Allineamento finale	G A T C G A T G A T - G A T

E in base al numero di sequenze:

- **Pairwise Alignment:** confronto tra due sequenze.
- **Multiple Sequence Alignment:** confronto tra più sequenze.

1.4 Algoritmi di Allineamento Classici

Gli algoritmi di allineamento delle sequenze, come **Needleman-Wunsch**, **Smith-Waterman** e **Hirschberg**, sono stati sviluppati per affrontare questa sfida computazionale, offrendo strumenti per dedurre allineamenti globali o locali ottimali in base a diverse esigenze analitiche.
Negli ultimi anni, la crescente disponibilità di dati sequenziati ha posto nuove sfide in termini di **efficienza computazionale** e **accuratezza** degli allineamenti, ciò ha stimolato lo sviluppo di nuove metodologie che migliorano o modificano gli algoritmi classici, integrando avanzamenti tecnologici come l'**intelligenza artificiale**, l'**apprendimento automatico** e l'utilizzo della **GPU**.

Capitolo 2

Approcci Algoritmici negli Algoritmi di Allineamento di Sequenze: Programmazione Dinamica e Divide et Impera

La **programmazione dinamica** e il metodo **divide et impera** [2] sono due paradigmi fondamentali nel campo degli algoritmi di allineamento di sequenze, ognuno con le sue peculiarità e applicazioni specifiche. Sebbene entrambi gli approcci mirino a decomporre problemi complessi in componenti più gestibili, lo fanno in modi che influenzano significativamente l'efficienza e la complessità dell'algoritmo risultante.

2.1 Programmazione Dinamica

La programmazione dinamica è particolarmente efficace negli algoritmi di allineamento di sequenze a causa della sua capacità di sfruttare la sovrapposizione dei sottoproblemi. Gli algoritmi come **Needleman-Wunsch** per l'allineamento globale e **Smith-Waterman** per l'allineamento locale utilizzano questa tecnica per costruire una matrice di punteggio, dove ogni cella $F(i, j)$ rappresenta il punteggio massimo ottenibile allineando i primi i caratteri di una sequenza con i primi j di un'altra.
Ogni cella della matrice è calcolata basandosi sui risultati già ottenuti dalle celle adiacenti, garantendo che ogni sottoproblema sia risolto una sola volta e i suoi risultati memorizzati per usi futuri. Questo approccio non solo riduce il tempo di calcolo ma anche minimizza il ricalcolo ridondante, una strategia nota come **memoizzazione**.
Nel contesto della programmazione dinamica per l'allineamento di sequenze, il calcolo è basato su una matrice bidimensionale in cui ogni valore di cella dipende dai tre valori:

1. **Sinistra (L):** valore della cella $(i, j - 1)$,

2. **Alto (U):** valore della cella $(i-1, j)$,

3. **Diagonale (D):** valore della cella $(i-1, j-1)$.

2.2 Divide et Impera

Il metodo **divide et impera**, d'altra parte, è utilizzato per gestire diversamente la complessità computazionale, soprattutto in termini di uso della memoria. Un esempio prominente nel contesto degli algoritmi di allineamento di sequenze è l'**algoritmo di Hirschberg**, che è una variante dell'algoritmo di Needleman-Wunsch ottimizzata per utilizzare meno memoria.
Questo algoritmo divide il problema dell'allineamento in due metà fino a che le sequenze non sono ridotte a lunghezze gestibili. Risolve poi questi sottoproblemi indipendentemente e ricombina i risultati per ottenere l'allineamento finale. L'approccio divide et impera è particolarmente vantaggioso per l'allineamento di sequenze molto lunghe dove l'impronta di memoria di una matrice completa sarebbe proibitiva.
In combinazione, la programmazione dinamica e il divide et impera offrono un set di strumenti potenti per affrontare una delle sfide più impegnative della bioinformatica: l'allineamento efficiente e accurato delle sequenze biologiche. Mentre la programmazione dinamica si concentra sull'ottimizzazione del tempo di calcolo tramite la memoizzazione, il divide et impera affronta il problema dal punto di vista della riduzione dello spazio, dimostrando che diversi problemi richiedono strategie su misura a seconda delle risorse e delle esigenze specifiche.

Entrambi gli approcci sono quindi essenziali per i bioinformatici che lavorano con database di sequenze sempre più grandi, dove sia la velocità che l'economia di memoria sono cruciali per analisi pratiche e tempestive.

Capitolo 3

Algoritmo di Needleman-Wunsch

L'**Algoritmo di Needleman-Wunsch** [4] rappresenta uno dei pilastri fondamentali nel campo dell'allineamento di sequenze, essendo stato specificatamente progettato per ottimizzare l'allineamento globale tra due sequenze biologiche, quali DNA, RNA o proteine. Questo metodo si distingue per l'uso efficace della programmazione dinamica per gestire e risolvere le complessità associate a tali confronti molecolari. L' allineamento globale implica che entrambe le sequenze vengano considerate per intero, anche se ciò comporta l'introduzione di gap.

3.1 Costruzione della Matrice di Costo

[3] Consideriamo due sequenze $X =$ "ACG" e $Y =$ "AG" per illustrare passaggio per passaggio la costruzione della matrice di costo usando l'algoritmo di Needleman-Wunsch.

3.2 Formula di Ricorrenza

La potenza dell'algoritmo di Needleman-Wunsch risiede nella sua formula di ricorrenza, che stabilisce come ciascun elemento della matrice di allineamento F sia derivato:

$$F(i,j) = \max \begin{cases} F(i-1, j-1) + S(X[i], Y[j]), \\ F(i-1, j) + \text{gap penalty}, \\ F(i, j-1) + \text{gap penalty} \end{cases} \tag{3.1}$$

dove :

- ogni cella $F(i,j)$ rappresenta il punteggio massimo ottenuto allineando i primi i caratteri di X con i primi j caratteri di Y
- $S(X[i], Y[j])$ rappresenta il punteggio assegnato per un match o mismatch tra i caratteri $X[i]$ e $Y[j]$
- **gap penalty** è la penalità imposta per l'inserimento di un gap in una delle sequenze

3.3 Inizializzazione della Matrice

La matrice viene inizializzata con $F(0,0) = 0$. L'inizializzazione della prima riga e della prima colonna va a riflettere le penalità di gap necessarie per allineare i caratteri di una sequenza con gap nell'altra:

	-	A	G
-	0	-1	-2
A	-1		
C	-2		
G	-3		

3.4 Popolamento della Matrice

Procediamo ora a riempire la matrice utilizzando la formula di ricorrenza. Analizziamo il primo passo significativo per il calcolo di $F(1,1)$:

	-	A	G
-	0	-1	-2
A	-1	1	
C	-2		
G	-3		

Il valore di $F(1,1)$ è calcolato come:

$$\max \left\{ \begin{array}{l} F(0,0) + S(A,A) = 0 + 1 = 1, \\ F(0,1) + \text{gap penalty} = -1 - 1 = -2, \\ F(1,0) + \text{gap penalty} = -1 - 1 = -2 \end{array} \right\} = 1$$

Dopo aver calcolato tutti i valori tramite la formula di ricorrenza, la matrice di costo completa per l'allineamento delle sequenze $X =$ "ACG" e $Y =$ "AG" è la seguente:

	-	A	G
-	0	-1	-2
A	-1	1	0
C	-2	0	0
G	-3	0	1

Il percorso di allineamento ottimale può essere determinato tramite backtracking dalla cella $F(n,m)$.

3.5 Backtracking per l'Allineamento Ottimale

Una volta che la matrice è completamente popolata, si procede con il **backtracking** per ottenere l'allineamento ottimale :

- **Inizio :** Partiamo dalla cella $F(n, m)$ il cui valore è il punteggio dell'allineamento ottimale delle intere sequenze.
- **Decisione di movimento :**
  - Se il punteggio in $F(i, j)$ deriva da $F(i-1, j-1) + S(X[i], Y[j])$, significa che $X[i]$ e $Y[j]$ sono stati allineati. Il percorso di backtracking procede diagonalmente.
  - Se il punteggio deriva da $F(i-1, j)$ + gap penalty, indica che $X[i]$ è stato allineato con un gap in Y. Il percorso procede verticalmente.
  - Se il punteggio viene da $F(i, j-1)$ + gap penalty, $Y[j]$ è allineato con un gap in X. Il percorso procede orizzontalmente.
- **Ripetizione :** Questo processo viene ripetuto, muovendosi di cella in cella secondo le decisioni sopra descritte, fino a raggiungere $F(0, 0)$
- **Risultato :** Se ci si muove diagonalmente, i caratteri corrispondenti vengono allineati. Se il movimento è verticale o orizzontale, si introduce un gap nella sequenza corrispondente.

3.6 Esempio Step-by-Step: Needleman-Wunsch

Consideriamo due sequenze biologiche per illustrare l'algoritmo di Needleman-Wunsch:

- **Sequenza 1 (X)**: GATTACA
- **Sequenza 2 (Y)**: GCATGCU

3.7 Punteggi Assegnati

- **Match**: +1
- **Mismatch**: -1
- **Gap Penalty**: -2

3.8 Inizializzazione della Matrice

La matrice di punteggio viene inizializzata con penalità di gap cumulative nella prima riga e nella prima colonna, rispettando la seguente struttura:

	$-$	G	C	A	T	G	C	U
$-$	0	-2	-4	-6	-8	-10	-12	-14
G	-2							
A	-4							
T	-6							
T	-8							
A	-10							
C	-12							
A	-14							

3.9 Riempimento della Matrice

Il riempimento della matrice avviene seguendo la formula di ricorrenza, considerando il massimo tra le seguenti opzioni:

- Diagonale: $F(i-1, j-1) + S(X[i], Y[j])$ (match o mismatch),
- Sopra: $F(i-1, j)$ + gap penalty (gap nella sequenza 2),
- Sinistra: $F(i, j-1)$ + gap penalty (gap nella sequenza 1).

Esempio di Calcolo:

- **Cella (1,1)**: Confronto G con G.

$$F(1,1) = \max\{F(0,0) + 1, F(0,1) - 2, F(1,0) - 2\} = \max\{1, -4, -4\} = 1$$

- **Cella (2,2)**: Confronto A con C.

$$F(2,2) = \max\{F(1,1) - 1, F(1,2) - 2, F(2,1) - 2\} = \max\{0, -1, -3\} = 0$$

3.10 Matrice Completata

Dopo aver completato tutti i calcoli, la matrice risulta piena:

	$-$	G	C	A	T	G	C	U
$-$	0	-2	-4	-6	-8	-10	-12	-14
G	-2	1	-1	-3	-5	-7	-9	-11
A	-4	-1	0	2	0	-2	-4	-6
T	-6	-3	-2	0	3	1	-1	-3
T	-8	-5	-4	-2	1	4	2	0
A	-10	-7	-3	-1	-1	2	3	1
C	-12	-9	-5	-3	-1	0	5	3
A	-14	-11	-7	-3	-2	-2	3	4

3.11 Backtracking per l'Allineamento Ottimale

Partendo dalla cella $F(7, 8) = 4$, risaliamo la matrice seguendo il percorso che massimizza il punteggio:

- Dalla cella $F(7, 8)$ a $F(6, 7)$ (match tra A e U, gap in X).
- Dalla cella $F(6, 7)$ a $F(5, 6)$ (match tra G e G).
- Si procede ripetendo i passi fino a raggiungere la cella $F(0, 0)$.

L'allineamento ottimale risultante è:

- **Sequenza 1 (X)**: G-ATTACA
- **Sequenza 2 (Y)**: GCA-TGCU

3.12 Applicazioni Pratiche

L'algoritmo di Needleman-Wunsch è utilizzato principalmente per:

- **Allineamenti globali di sequenze**: Ideale per sequenze di lunghezza simile, come geni omologhi o sequenze proteiche appartenenti alla stessa famiglia.
- **Analisi di varianti genomiche**: Utilizzato per confrontare varianti di geni o segmenti di DNA di organismi correlati, fornendo un allineamento che copre l'intera lunghezza delle sequenze.

Grazie all'allineamento globale, l'algoritmo di Needleman-Wunsch permette di confrontare sequenze su larga scala, mantenendo intatta la loro struttura complessiva.

Capitolo 4

Algoritmo di Smith-Waterman

L'algoritmo di Smith-Waterman [7] rappresenta un approccio cruciale per l'allineamento locale di sequenze biologiche, come DNA, RNA e proteine. A differenza dell'allineamento globale, che considera l'intera lunghezza delle sequenze, l'allineamento locale si concentra su segmenti altamente simili, ignorando le regioni meno rilevanti. Questa caratteristica lo rende particolarmente utile per identificare motivi conservati o regioni omologhe, anche se le sequenze complete non sono correlate.
L'algoritmo sfrutta una matrice di punteggio per valutare le possibili combinazioni di allineamento tra due sequenze. A differenza di Needleman-Wunsch, introduce una peculiarità fondamentale: il valore di una cella può essere resettato a zero, permettendo di ignorare porzioni che non migliorano il punteggio. Questo consente di focalizzarsi esclusivamente sulle regioni di interesse.

4.1 Formula di Ricorrenza

La formula di ricorrenza dell'algoritmo di Smith-Waterman si presenta nel seguente modo:

$$F(i,j) = \max \left\{ \begin{array}{l} 0, \\ F(i-1,j-1) + S(X[i], Y[j]), \\ F(i-1,j) + \text{gap penalty}, \\ F(i,j-1) + \text{gap penalty} \end{array} \right\}$$

Ogni cella $F(i,j)$ rappresenta il punteggio massimo ottenibile allineando i primi i caratteri della sequenza X con i primi j caratteri della sequenza Y. Le componenti della formula possono essere interpretate come segue:

- **0**: Il reset a zero consente di ignorare segmenti non rilevanti.

- **Match/Mismatch**: $F(i-1, j-1) + S(X[i], Y[j])$, dove $S(X[i], Y[j])$ è il punteggio assegnato per un match o mismatch tra i caratteri.
- **Gap Penalty**: Penalità applicata per l'inserimento di un gap in una delle due sequenze.

L'inclusione del valore zero nella formula è il tratto distintivo dell'algoritmo, permettendo di interrompere un allineamento quando non contribuisce al risultato ottimale.

4.2 Esempio: Costruzione della Matrice

Consideriamo l'allineamento locale tra due sequenze: $X =$ "ACTA" e $Y =$ "ACA". Usiamo i seguenti punteggi:

- **Match**: +2
- **Mismatch**: -1
- **Gap Penalty**: -2

La matrice iniziale è impostata con tutti i valori pari a zero nella prima riga e nella prima colonna. Procediamo passo per passo:

Step 1: Calcolo di $F(1,1)$ Confrontiamo $X[1] = A$ e $Y[1] = A$:

$$F(1,1) = \max \left\{ \begin{array}{l} 0, \\ F(0,0) + 2 = 2, \\ F(0,1) - 2 = -2, \\ F(1,0) - 2 = -2 \end{array} \right\} = 2$$

Step 2: Calcolo di $F(1,2)$ Confrontiamo $X[1] = A$ e $Y[2] = C$:

$$F(1,2) = \max \left\{ \begin{array}{l} 0, \\ F(0,1) - 1 = -1, \\ F(0,2) - 2 = -2, \\ F(1,1) - 2 = 0 \end{array} \right\} = 0$$

Step 3: Calcolo di $F(2,3)$ Confrontiamo $X[2] = C$ e $Y[3] = A$:

$$F(2,3) = \max \left\{ \begin{array}{l} 0, \\ F(1,2) - 1 = -1, \\ F(1,3) - 2 = -2, \\ F(2,2) + 2 = 4 \end{array} \right\} = 4$$

La matrice completa risultante è:

	$-$	A	C	A
$-$	0	0	0	0
A	0	2	0	2
C	0	0	4	2
T	0	0	2	3
A	0	2	0	4

4.3 Backtracking

Una volta completata la matrice, si procede con il backtracking per determinare l'allineamento locale ottimale. Il punto di partenza è la cella con il valore massimo nella matrice, che rappresenta il segmento di massima similarità tra le due sequenze.
Nel nostro esempio, il valore massimo si trova nella cella $F(4,4) = 4$, che corrisponde a un match tra A e A. Da qui, si risale la matrice seguendo il percorso che ha portato al valore massimo, tenendo traccia dei caratteri allineati. Il risultato finale è:

- **Sequenza 1**: ACTA
- **Sequenza 2**: A-CA

Il backtracking termina quando si raggiunge una cella con valore zero, segnalando la fine del segmento allineato.

4.4 Applicazioni

L'algoritmo di Smith-Waterman è ampiamente utilizzato in bioinformatica per:

- Identificare regioni altamente simili tra sequenze, come motivi proteici conservati.
- Ricerca di somiglianze locali in grandi banche dati biologiche.
- Analisi di segmenti genomici specifici per studiare relazioni evolutive o funzionali.

Grazie alla sua capacità di focalizzarsi sulle regioni più rilevanti, Smith-Waterman rimane uno strumento fondamentale per l'analisi locale delle sequenze biologiche, nonostante la sua intensità computazionale.

Capitolo 5

Algoritmo di Hirschberg

5.1 Scopo e Applicazioni

L'algoritmo di Hirschberg [**?**] è una versione ottimizzata dell'algoritmo di Needleman-Wunsch per l'allineamento globale. La sua progettazione mira a ridurre drasticamente il consumo di memoria, rendendolo particolarmente utile per l'allineamento di sequenze molto lunghe. Questo lo rende prezioso in contesti bioinformatici su larga scala, come il confronto di sequenze genomiche complete.
Mentre Needleman-Wunsch richiede una matrice completa di dimensione $O(n \cdot m)$, Hirschberg utilizza un approccio *divide-et-impera* per ridurre la complessità spaziale a $O(n + m)$. Questo è fondamentale quando si confrontano sequenze che non possono essere gestite interamente in memoria.

5.2 Funzionamento

L'algoritmo di Hirschberg suddivide il problema di allineamento globale in sottoproblemi più piccoli, risolvendoli ricorsivamente e combinandone i risultati. Il funzionamento si basa sui seguenti passi:

1. **Calcolo dei punteggi della matrice dimezzata**: anziché calcolare l'intera matrice, Hirschberg memorizza solo due righe alla volta, corrispondenti allo stato attuale e al precedente. Questo riduce il consumo di memoria senza sacrificare accuratezza.

2. **Individuazione del punto di divisione ottimale**: divide le sequenze nel punto in cui la somma dei punteggi delle due metà è massimizzata, determinando il punto centrale ottimale.

3. **Ricorsione e concatenazione dei risultati**: calcola ricorsivamente l'allineamento delle sottosequenze divise e combina i risultati per ottenere l'allineamento globale finale.

Grazie a questa strategia, Hirschberg mantiene la stessa complessità temporale di Needleman-Wunsch, $O(n \cdot m)$, ma con un consumo di memoria significativamente ridotto.

5.3 Esempio Step-by-Step

Per illustrare il funzionamento dell'algoritmo di Hirschberg, consideriamo l'allineamento globale tra le seguenti sequenze:

- **Sequenza 1**: ACACACTA
- **Sequenza 2**: AGCACACA

Passo 1: Divisione delle Sequenze

Dividiamo la sequenza 1 a metà, ottenendo:

- **Prima metà**: ACACA
- **Seconda metà**: CTA

L'obiettivo è calcolare la metà sinistra della matrice di Needleman-Wunsch per la prima metà della Sequenza 1 e l'intera Sequenza 2, memorizzando solo due righe alla volta.

Passo 2: Calcolo dei Punteggi

Applichiamo Needleman-Wunsch riducendo la memoria. La matrice risultante è la seguente:

	$-$	A	G	C	A	C	A	C	A
$-$	0	-2	-4	-6	-8	-10	-12	-14	-16
A	-2	2	0	-2	-4	-6	-8	-10	-12
C	-4	0	1	3	1	4	2	5	3
A	-6	2	-1	1	5	3	7	5	9
C	-8	0	1	3	3	7	5	8	6
A	-10	2	-1	1	5	5	9	7	10

Il punto di divisione ottimale viene scelto analizzando la colonna centrale della matrice, che divide Sequenza 2 in due metà.

Passo 3: Ricorsione

Dividiamo Sequenza 2 in:

- **Prima metà**: AGC
- **Seconda metà**: ACACA

Calcoliamo l'allineamento della prima metà delle sequenze e, successivamente, della seconda.

Esempio di Calcolo per la Prima Metà Per la prima metà ACACA e AGC, utilizziamo Needleman-Wunsch. La matrice calcolata è:

	$-$	A	G	C
$-$	0	-2	-4	-6
A	-2	2	0	-2
C	-4	0	1	3
A	-6	2	-1	1
C	-8	0	1	3
A	-10	2	-1	1

Il backtracking produce:

- **Sequenza 1 (prima metà)**: ACACA
- **Sequenza 2 (prima metà)**: A-GC

Esempio di Calcolo per la Seconda Metà Per la seconda metà CTA e ACACA, la matrice risultante è:

	$-$	A	C	A	C	A
$-$	0	-2	-4	-6	-8	-10
C	-2	-2	2	0	4	2
T	-4	-4	0	1	2	0
A	-6	-2	-2	3	1	5

Il backtracking produce:

- **Sequenza 1 (seconda metà)**: -CTA
- **Sequenza 2 (seconda metà)**: ACACA

Passo 4: Concatenazione dei Risultati

Unendo i risultati delle due metà, otteniamo:

- **Sequenza 1**: ACACA-CTA
- **Sequenza 2**: A-GCACACA

5.4 Applicazioni

L'algoritmo di Hirschberg trova applicazione in vari ambiti della bioinformatica:

- **Allineamento globale di sequenze genomiche**: ideale per confrontare interi genomi o lunghe sequenze dove l'efficienza in memoria è critica.
- **Confronto di geni omologhi**: utilizzato per analizzare relazioni evolutive tra specie.
- **Studi su larga scala**: supporta analisi su dataset biologici complessi, garantendo accuratezza ed efficienza.

Grazie alla sua ridotta complessità spaziale, Hirschberg rappresenta una soluzione ottimale per l'allineamento di sequenze lunghe, superando i limiti di memoria imposti da algoritmi tradizionali come Needleman-Wunsch.

Capitolo 6

Gestione delle Penalità per i Gap negli Algoritmi di Allineamento

6.1 Penalità per i Gap

Nell'allineamento delle sequenze, l'introduzione di gap (inserzioni o delezioni) è spesso necessaria per massimizzare la similarità tra le sequenze. Tuttavia, l'inserimento di gap comporta una penalità nel calcolo del punteggio di allineamento. Esistono diversi modelli per assegnare queste penalità:

- **Penalità Lineare**: Ogni gap, indipendentemente dalla sua lunghezza, riceve una penalità costante. Questo approccio è semplice ma non riflette accuratamente le dinamiche biologiche, poiché l'apertura di un nuovo gap è generalmente meno probabile dell'estensione di un gap esistente.
- **Penalità Affine**: Questo modello introduce una penalità maggiore per l'apertura di un gap (ρ) e una penalità minore per l'estensione del gap (σ). La penalità totale per un gap di lunghezza k è calcolata come $\rho + (k-1) \cdot \sigma$. Questo approccio è più realistico dal punto di vista biologico, poiché riflette il costo maggiore associato all'inizio di un gap rispetto alla sua continuazione.

6.2 Algoritmo di Needleman-Wunsch con Penalità Affine

L'algoritmo di Needleman-Wunsch può essere esteso per incorporare penalità affini. In questo caso, si utilizzano tre matrici:

- $M(i,j)$: Punteggio massimo per l'allineamento delle sottosequenze $X[1..i]$ e $Y[1..j]$ senza terminare con un gap.
- $X(i,j)$: Punteggio massimo per l'allineamento che termina con un gap nella sequenza X.

- $Y(i, j)$: Punteggio massimo per l'allineamento che termina con un gap nella sequenza Y.

Le relazioni di ricorrenza sono le seguenti:

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

$$X(i,j) = \max \begin{cases} M(i-1,j) - \rho \\ X(i-1,j) - \sigma \end{cases}$$

$$Y(i,j) = \max \begin{cases} M(i,j-1) - \rho \\ Y(i,j-1) - \sigma \end{cases}$$

Dove $S(x_i, y_j)$ è il punteggio per l'allineamento dei caratteri x_i e y_j. Il punteggio ottimale finale è dato da $\max(M(n,m), X(n,m), Y(n,m))$, dove n e m sono le lunghezze delle sequenze X e Y rispettivamente.

6.3 Esempio Pratico

Consideriamo due sequenze proteiche:

```
Sequenza 1: ACGTACGT
Sequenza 2: ACG--CGT
```

Supponiamo di utilizzare:

- **Punteggio di match**: +2
- **Punteggio di mismatch**: -1
- **Penalità di apertura del gap** (ρ): -3
- **Penalità di estensione del gap** (σ): -1

Con penalità affini, il punteggio finale è $+8$, un valore più accurato biologicamente.

6.4 Conclusioni

Le penalità affini per i gap rappresentano un miglioramento significativo rispetto alla penalità lineare, poiché modellano in modo più accurato le dinamiche di mutazione e di evoluzione biologica. Incorporare queste penalità negli algoritmi di allineamento migliora la qualità degli allineamenti ottenuti, in particolare per le sequenze proteiche, dove inserzioni o delezioni tendono a formare blocchi. Questo approfondimento sulle penalità affini è un aspetto cruciale per un allineamento più realistico e applicabile in contesti biologici.

Capitolo 7

Confronto delle Complessità Spaziali e Temporali degli Algoritmi

Il confronto delle complessità spaziali e temporali dei tre algoritmi classici di allineamento delle sequenze — **Needleman-Wunsch**, **Smith-Waterman** e **Hirschberg** — è fondamentale per comprendere i contesti applicativi in cui ciascun algoritmo risulta vantaggioso. La seguente tabella riassume le complessità computazionali di ciascun algoritmo.

Algoritmo	**Complessità Temporale**	**Complessità Spaziale**	**Note**
Needleman-Wunsch	$O(n \cdot m)$	$O(n \cdot m)$	Adatto per allineamenti globali.
Smith-Waterman	$O(n \cdot m)$	$O(n \cdot m)$	Ottimale per allineamenti locali, ma richiede molta memoria.
Hirschberg	$O(n \cdot m)$	$O(n + m)$	Riduce il consumo di memoria per sequenze molto lunghe.

Tabella 7.1: Confronto delle complessità computazionali per Needleman-Wunsch, Smith-Waterman e Hirschberg.

7.1 Discussione delle Complessità

- **Needleman-Wunsch**: L'algoritmo di Needleman-Wunsch è efficace per gli allineamenti globali grazie alla sua capacità di confrontare due sequenze nella loro interezza. Tuttavia, la complessità spaziale $O(n \cdot m)$ rende il suo utilizzo impegnativo in termini di memoria per sequenze lunghe, come quelle genomiche complete.
- **Smith-Waterman**: Anche il Smith-Waterman presenta una complessità temporale e spaziale di $O(n \cdot m)$. Questo lo rende adatto all'allineamento locale, permettendo di identificare regioni simili senza considerare le porzioni meno rilevanti delle sequenze. Tuttavia, per sequenze molto lunghe, l'elevata complessità spaziale rappresenta un limite operativo, in quanto richiede grandi quantità di memoria per eseguire il calcolo della matrice.
- **Hirschberg**: L'algoritmo di Hirschberg è una variante ottimizzata del Needleman-Wunsch, progettata per ridurre il consumo di memoria. Pur mantenendo una complessità temporale

di $O(n \cdot m)$, sfrutta un approccio divide-et-impera che permette di ottenere una complessità spaziale di $O(n + m)$. Questo lo rende particolarmente vantaggioso per l'allineamento globale di sequenze molto lunghe, dove la memoria del sistema è una risorsa limitata.

7.2 Grafico della Complessità Spaziale

Il grafico seguente rappresenta visivamente come la complessità spaziale varia al crescere della lunghezza delle sequenze (n e m). Questo confronto evidenzia i benefici dell'approccio di Hirschberg rispetto a Needleman-Wunsch e Smith-Waterman quando si lavora con sequenze di grandi dimensioni.

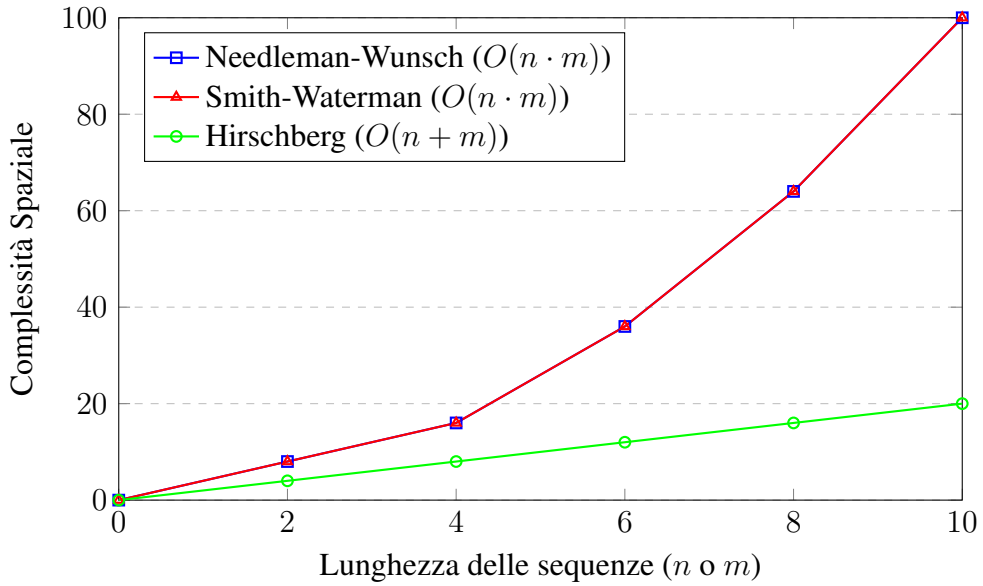

Figura 7.1: Confronto della crescita della complessità spaziale in base alla lunghezza delle sequenze.

La riduzione della complessità spaziale offerta dall'algoritmo di Hirschberg è particolarmente rilevante per applicazioni genomiche su larga scala, dove l'allineamento di interi genomi richiede il massimo risparmio di memoria.

Capitolo 8

Confronto tra Needleman-Wunsch e Hirschberg

8.1 Introduzione

Gli algoritmi di allineamento delle sequenze biologiche sono strumenti fondamentali in bioinformatica per il confronto e l'analisi di DNA, RNA e proteine. Questo capitolo esplora due algoritmi di allineamento globale: **Needleman-Wunsch** e **Hirschberg**. Entrambi permettono di trovare il miglior allineamento tra due sequenze, ma si differenziano significativamente per quanto riguarda il consumo di memoria e il tempo di esecuzione.
Per l'analisi e i test, sono state utilizzate implementazioni disponibili nella repository GitHub `https://github.com/annabellej/cs466_hirschberg`, che fornisce il codice sorgente e strumenti per la profilazione delle prestazioni. In questo capitolo, descriviamo il funzionamento del codice, i risultati ottenuti e le differenze chiave tra i due algoritmi.

8.2 Codice Utilizzato

Il codice testato implementa entrambi gli algoritmi e include funzioni per la generazione di sequenze casuali, la profilazione della memoria e del tempo, e il confronto dei risultati. Di seguito, analizziamo le componenti principali del codice.

8.2.1 Needleman-Wunsch

L'algoritmo di Needleman-Wunsch, implementato nella funzione `global_align`, utilizza una matrice bidimensionale per calcolare il punteggio massimo di allineamento tra due sequenze. Una matrice di puntatori consente di ricostruire l'allineamento ottimale tramite il *traceback*.

Funzionamento Il calcolo avviene iterando sulle due sequenze, confrontando caratteri corrispondenti e aggiungendo penalità per gap o mismatch. Alla fine, il *traceback* costruisce le sequenze allineate.

Esempio di utilizzo

```
keys = ['A', 'C', 'T', 'G', '-']
delta = {}
for i in range(len(keys)):
    delta[keys[i]] = {k: (1 if keys[i] == k else -1) for k in keys}

score, alignment = global_align("TAGATA", "GTAGGCTTAAGGTTA", delta)
print(score, alignment)
```

Risultato

- **Punteggio:** `-3`
- **Allineamento:**

```
-TA-G----A---TA
GTAGGCTTAAGGTTA
```

8.2.2 Hirschberg

L'algoritmo di Hirschberg, implementato nella funzione `hirschberg`, rappresenta un miglioramento in termini di efficienza della memoria. Utilizza un approccio ricorsivo per dividere il problema in sottoproblemi più piccoli, evitando di costruire l'intera matrice.

Funzionamento Hirschberg calcola una colonna della matrice alla volta, riducendo il consumo di memoria. Utilizzando un approccio *divide et impera*, l'algoritmo riduce il problema a metà fino a raggiungere sottoproblemi semplici da risolvere.

Esempio di utilizzo

```
alignment = hirschberg(0, 0, len("ATGTC"), len("ATCGC"), None, "ATGTC", "ATCGC")
print(alignment)
```

Risultato

- **Allineamento:**

```
AT-GTC
ATCG-C
```

8.2.3 Generazione di Sequenze Casuali

Il codice include una funzione per generare sequenze casuali, utilizzata per testare gli algoritmi su scale diverse:

```
def genSequence(n):
    return ''.join(random.choice('ACGT') for _ in range(n))

dna_sequences = {}
for i in range(1, 1100, 40):
    dnasequences[i] = [(genSequence(i), genSequence(i)) for  in range(5)]
```

Questa funzione genera coppie di sequenze con lunghezze da 1 a 1100 (incrementi di 40), ognuna ripetuta 5 volte per garantire risultati affidabili.

8.2.4 Profilazione della Memoria e del Tempo

Per ogni algoritmo, la funzione `profile_memory_and_time` monitora il consumo di memoria e il tempo di esecuzione:

```
fig = profile_memory_and_time(global_align, "TAGATA", "GTAGGCTTAAGGTTA", delta)
fig.show()
```

Risultati

- **Needleman-Wunsch:**
  - **Memoria:** `3.5 MiB`
  - **Tempo:** `0.24 secondi`
- **Hirschberg:**
  - **Memoria:** `1.0 MiB`
  - **Tempo:** `0.28 secondi`

8.3 Risultati dei Test

8.3.1 Uso della Memoria nel Tempo

I grafici mostrano come il consumo di memoria di Needleman-Wunsch cresca gradualmente fino a un picco massimo durante l'esecuzione, mentre Hirschberg mantiene un consumo stabile e significativamente inferiore.

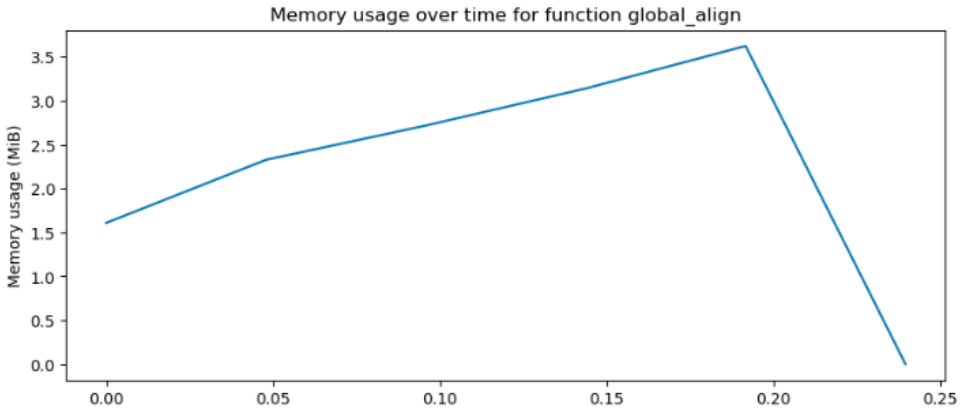

Figura 8.1: Uso della memoria nel tempo per Needleman-Wunsch. Il grafico mostra un consumo crescente fino a un picco massimo durante l'esecuzione, dovuto alla costruzione della matrice completa.

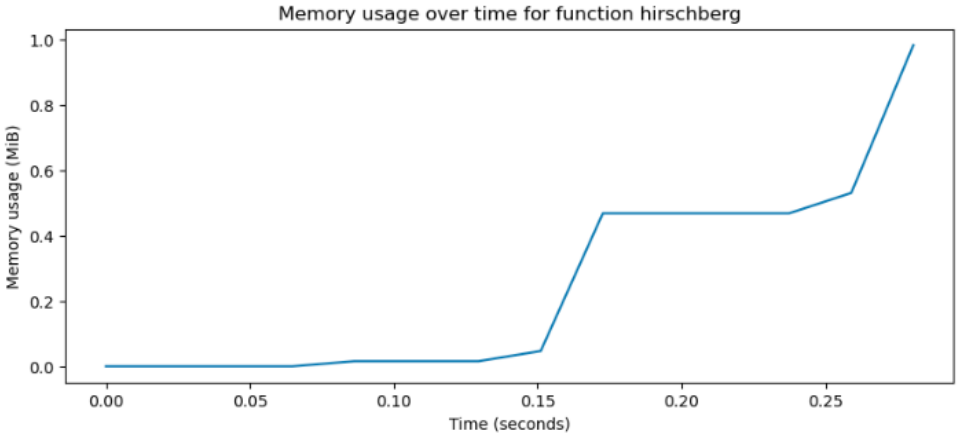

Figura 8.2: Uso della memoria nel tempo per Hirschberg. Il consumo di memoria è più basso e stabile grazie all'approccio ricorsivo che evita la costruzione di una matrice completa.

8.3.2 Memoria e Tempo rispetto alla Lunghezza delle Sequenze

- **Memoria:** Needleman-Wunsch cresce proporzionalmente alla lunghezza delle sequenze, mentre Hirschberg rimane quasi costante.
- **Tempo:** Entrambi gli algoritmi mostrano un tempo crescente con la lunghezza delle sequenze. Needleman-Wunsch è leggermente più veloce per sequenze brevi, ma la differenza si riduce con l'aumentare della lunghezza.

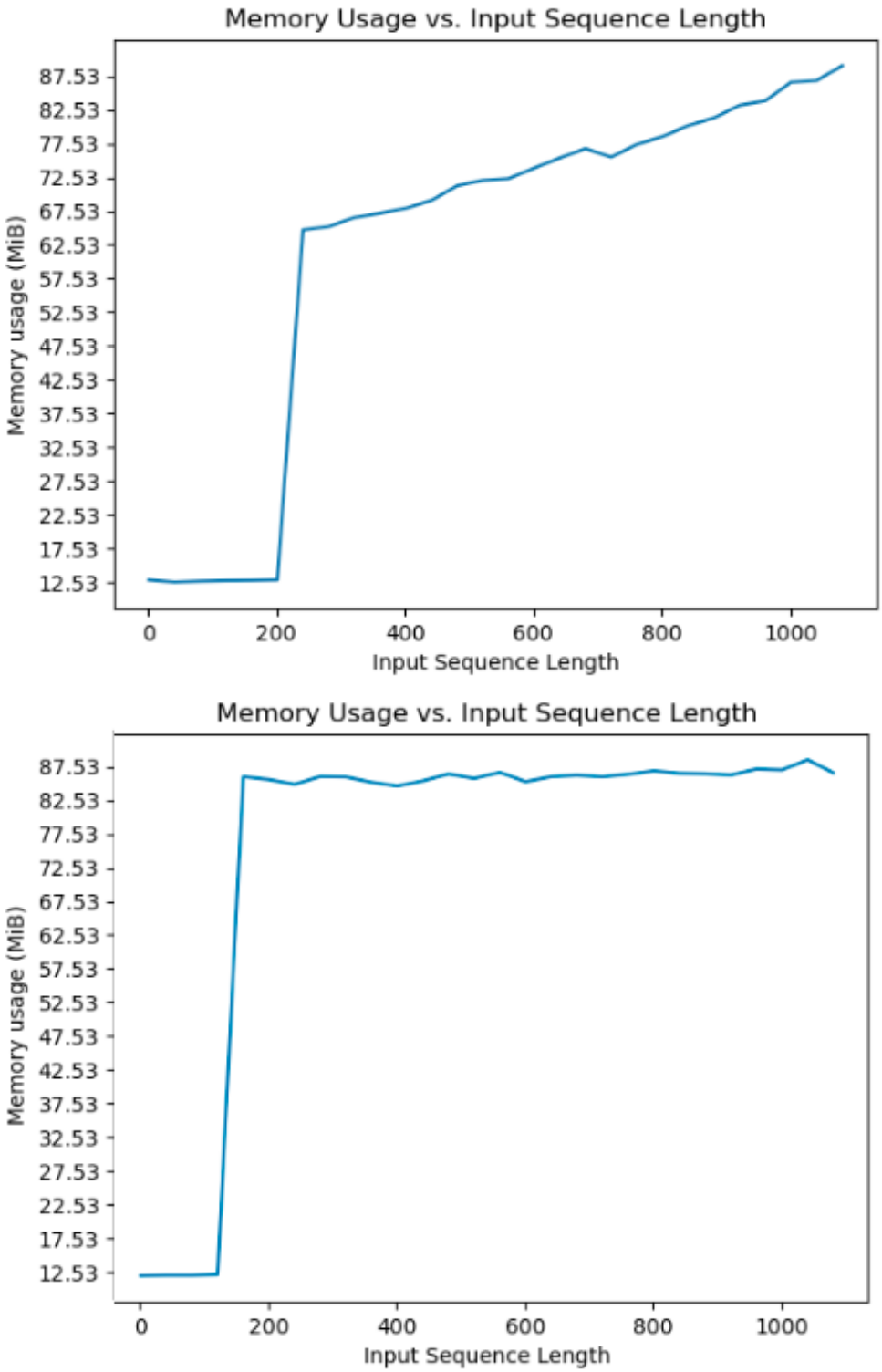

Figura 8.3: Confronto dell'uso della memoria rispetto alla lunghezza delle sequenze. Needleman-Wunsch mostra un consumo crescente quasi lineare, mentre Hirschberg mantiene un consumo costante.

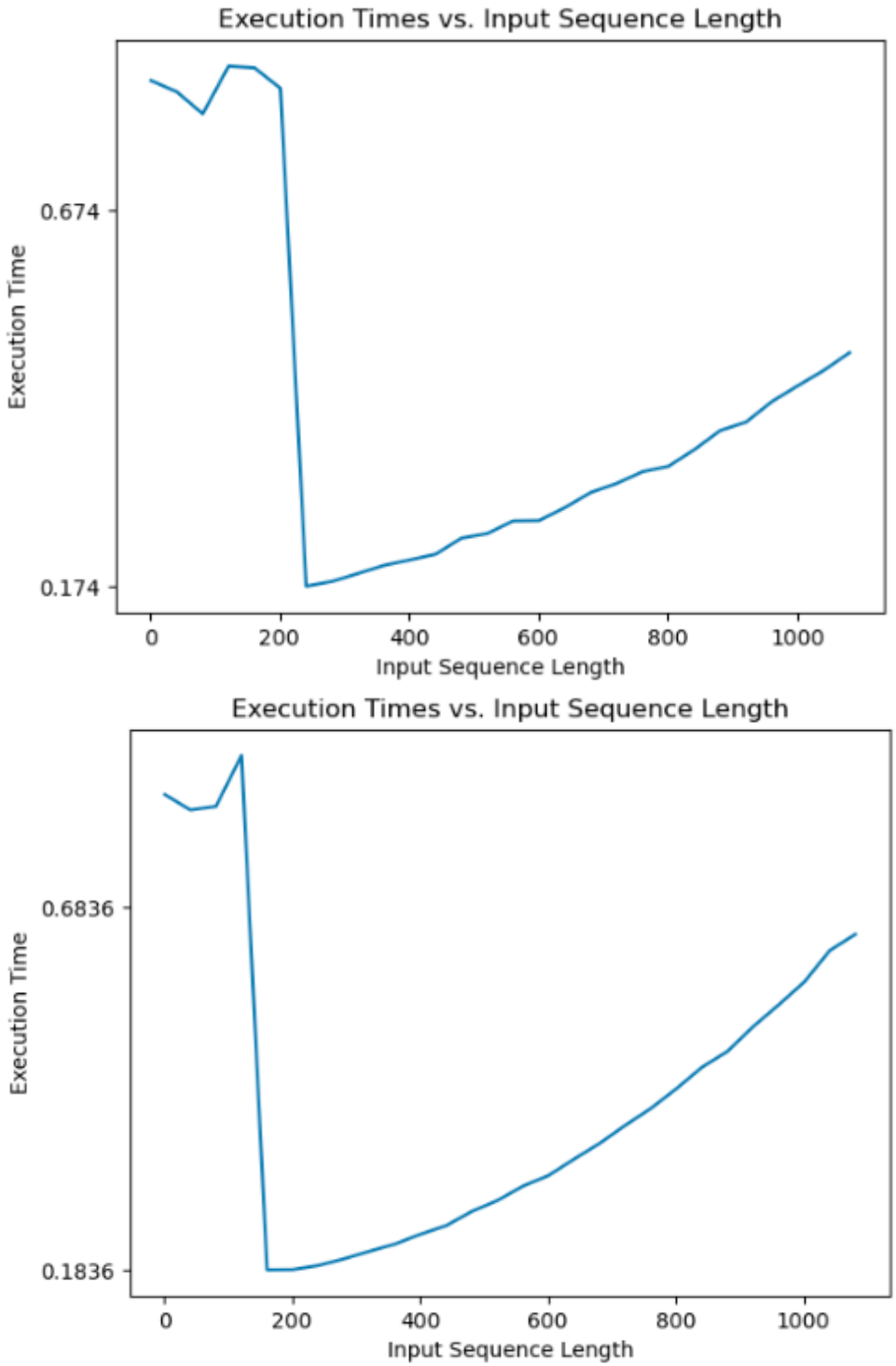

Figura 8.4: Confronto del tempo di esecuzione rispetto alla lunghezza delle sequenze. Entrambi gli algoritmi mostrano un tempo crescente, ma Needleman-Wunsch è leggermente più veloce per sequenze brevi.

8.4 Discussione

Efficienza Spaziale: Hirschberg si distingue per il suo basso consumo di memoria, rendendolo adatto per sequenze lunghe o ambienti con risorse limitate. Needleman-Wunsch, invece, consuma più memoria, ma offre una maggiore semplicità di implementazione.
Efficienza Temporale: Needleman-Wunsch è generalmente più veloce per sequenze corte, ma Hirschberg diventa comparabile man mano che le sequenze aumentano di lunghezza.

8.5 Conclusioni

Questo confronto dimostra che entrambi gli algoritmi hanno punti di forza e debolezze. Needleman-Wunsch è ideale per analisi rapide su sequenze brevi, mentre Hirschberg è la scelta migliore per scenari in cui la memoria rappresenta un vincolo critico. Entrambi offrono soluzioni complementari per applicazioni bioinformatiche diverse.

Capitolo 9

Approccio Parallelo su GPU per l'Allineamento di Sequenze

9.1 Importanza degli Acceleratori Hardware

Con la crescente disponibilità di dati biologici di grandi dimensioni, come i dataset derivati dalle tecnologie di sequenziamento long-read, l'efficienza degli algoritmi di allineamento è diventata una sfida cruciale in bioinformatica. Algoritmi classici come *Needleman-Wunsch* e *Smith-Waterman* soffrono di limiti computazionali intrinseci, dovuti alla loro complessità temporale e spaziale quadratica. Gli acceleratori hardware, come GPU e IPU, offrono una soluzione moderna per affrontare queste sfide, migliorando sia la velocità che l'efficienza nell'elaborazione di dati complessi.

9.2 Utilizzo delle GPU

Negli ultimi anni, l'uso delle **Graphics Processing Units (GPU)** [6] è notevolmente aumentato per migliorare le prestazioni delle unità computazionali. Le GPU moderne sono in grado di gestire compiti estremamente complessi con miglioramenti significativi in termini di prestazioni.

- **Capacità computazionale:** Le GPU stanno emergendo come hardware potente rispetto alle CPU (*Central Processing Units*), grazie alla capacità di eseguire operazioni in parallelo.
- **Architettura CUDA:** L'interfaccia di programmazione *Compute Unified Device Architecture (CUDA)*, sviluppata da NVIDIA, consente di sfruttare le GPU non solo per applicazioni grafiche, ma anche per calcoli computazionali generici.

9.3 Metodi di Generazione della Matrice

I valori della matrice possono essere generati in diversi modi:

- **Riga per riga (row-wise):** Le celle sono calcolate sequenzialmente per riga.
- **Colonna per colonna (column-wise):** Le celle sono calcolate sequenzialmente per colonna.
- **Anti-diagonale (anti-diagonal-wise):** Le celle sono calcolate lungo anti-diagonali, consentendo un parallelismo elevato.

Tra questi, il metodo anti-diagonale è il più favorevole per il calcolo parallelo, come mostrato in Figura 9.1.

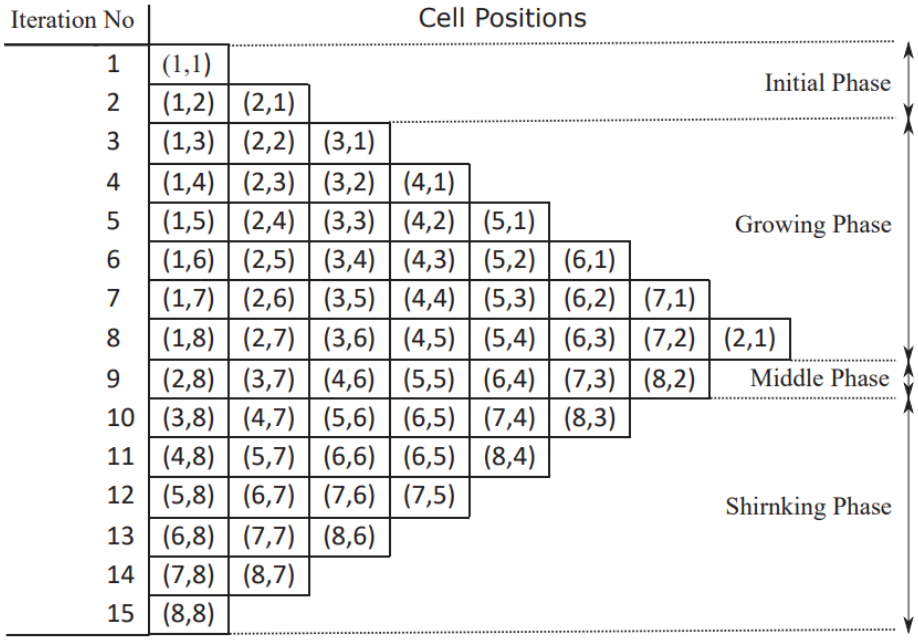

Figura 9.1: Generazione dei valori delle celle in modo anti-diagonale.

9.4 Fasi del Processo di Calcolo

Il calcolo della matrice è suddiviso in quattro fasi principali: inizializzazione, fase di crescita, fase centrale e fase di riduzione.

9.4.1 1. Initial Phase

Questa fase si occupa della prima e della seconda riga della matrice:

- Le celle della prima riga e della prima colonna vengono inizializzate con penalità di gap.

- Tutti gli altri valori intermedi sono calcolati utilizzando le due righe precedenti.

Esempio per una matrice 8×8:

- Iterazione 1: $(1, 1)$
- Iterazione 2: $(1, 2), (2, 1)$

9.4.2 2. Growing Phase

Dalla terza all'ottava riga, il numero di celle calcolate aumenta progressivamente. Per ogni cella (i, j), i valori sono calcolati considerando:

- **Diagonale:** $(i - 2, j - 1)$,
- **Sinistra:** $(i - 1, j - 1)$,
- **Sopra:** $(i - 2, j)$.

Esempio:

- Iterazione 3: $(1, 3), (2, 2), (3, 1)$
- Iterazione 4: $(1, 4), (2, 3), (3, 2), (4, 1)$

9.4.3 3. Middle Phase

Questa fase coinvolge la riga centrale della matrice (nona riga per una matrice 8×8). Le celle sono calcolate utilizzando:

- **Diagonale:** $(i - 2, j)$,
- **Sinistra:** $(i - 1, j - 2)$,
- **Sopra:** $(i - 1, j + 2)$.

Esempio:

- Iterazione 9: $(6, 4), (7, 3), (8, 2)$

9.4.4 4. Shrinking Phase

Dalla decima alla quindicesima riga, il numero di celle calcolate diminuisce gradualmente. Per ogni cella (i, j), i valori sono calcolati considerando:

- **Diagonale:** $(i - 2, j + 1)$,
- **Sinistra:** $(i - 1, j)$,

- **Sopra:** $(i-1, j+1)$.

Esempio:

- Iterazione 10: $(7, 4), (8, 3)$
- Iterazione 11: $(8, 4)$

Capitolo 10

Algoritmo Proposto per l'Allineamento di Sequenze

10.1 Inizializzazione delle Matrici FM e TM

L'algoritmo proposto utilizza due matrici principali per eseguire l'allineamento:

- **Foundation Matrix (FM):** Memorizza i punteggi calcolati in base al massimo valore tra tre celle adiacenti (diagonale, sinistra, sopra).
- **Traceback Matrix (TM):** Memorizza le direzioni di provenienza dei valori (diagonale, sinistra, sopra) per facilitare la ricostruzione del percorso di allineamento ottimale.

Ogni cella della FM è calcolata considerando i seguenti parametri:

- Penalità di *gap* (GV).
- Punteggio per il *match* (MV), inizializzato a 1.
- Penalità per il *mismatch* (SC), inizializzato a -1.

Le celle della prima riga e della prima colonna della FM sono inizializzate con valori di penalità di *gap*. La matrice TM registra i movimenti come:

- **U (up):** movimento verso l'alto.
- **L (left):** movimento verso sinistra.
- **D (diagonal):** movimento in diagonale.

10.2 Esempio di Inizializzazione

La Figura 10.1 mostra un esempio di inizializzazione delle matrici FM e TM per due sequenze:

- Query Sequence: **ACT**.
- Subject Sequence: **ACCA**.

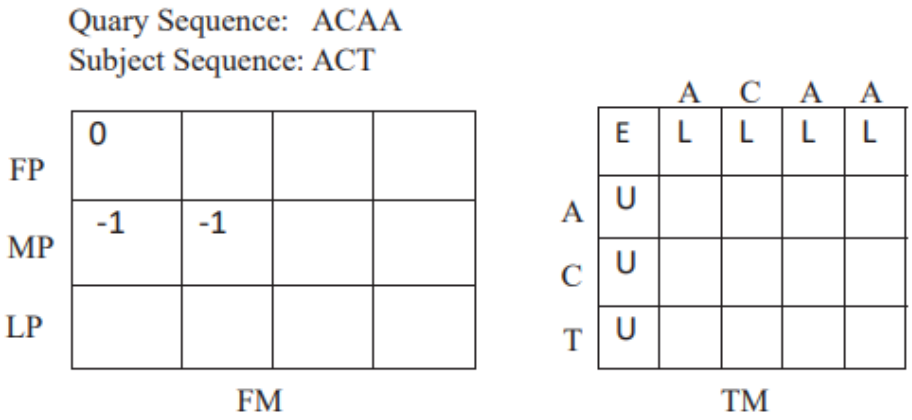

Figura 10.1: Esempio di inizializzazione delle matrici FM e TM.

—

10.3 Algoritmi Utilizzati

L'algoritmo proposto è suddiviso in due parti principali:

1. **Algoritmo 1: Metodo GPU-Based.**
2. **Algoritmo 2: Calcolo del Valore Massimo (MAX).**

10.3.1 Algoritmo 1: Metodo GPU-Based

L'algoritmo calcola l'allineamento globale ottimale tra due sequenze basandosi su una matrice $3 \times ((n \text{ o } m) + 1)$, dove n e m sono le lunghezze delle sequenze. La TM traccia il percorso ottimale attraverso la matrice.

```
Input: Two sequences S1 = a1, a2, ..., an and S2 = b1, b2, ..., bm
Output: Optimal alignment and score SC

FM: Foundation Matrix of size (n+1) × (m+1)
TM: Traceback Matrix
GV: Gap Value, MV: Match Value

1) Initialize FM(0,0) = 0 and TM(0,0) = 'E'
2) Fill first row and column with gap penalties
```

```
3) Loop through FM for i = 1 to n, j = 1 to m:
    a) Calculate FM(i, j) = max(FM(i-1, j-1) + MV,
                               FM(i-1, j) + GV,
                               FM(i, j-1) + GV)
    b) Record direction in TM(i, j)
4) Traceback to form alignment using TM
```

—

10.3.2 Algoritmo 2: Calcolo del Valore Massimo (MAX)

L'algoritmo calcola il massimo valore tra le tre direzioni:

- **D1:** Valore diagonale.
- **D2:** Valore sinistro.
- **D3:** Valore superiore.

```
Input: D1, D2, D3
Output: MAX value and pointer PTR

1) If D1 > max(D2, D3): MAX = D1, PTR = 'D'
2) Else if D2 > D3: MAX = D2, PTR = 'L'
3) Else: MAX = D3, PTR = 'U'
```

10.4 Allineamento Utilizzando la Matrice di Traceback (TM)

L'ultima fase della procedura di allineamento inizia subito dopo il completamento del riempimento della matrice FM. La **Traceback Matrix (TM)** è utilizzata per ricostruire le sequenze allineate, seguendo le direzioni memorizzate nella matrice durante la fase di calcolo.

- La TM ha dimensioni $(n+1) \times (m+1)$, dove n e m sono le lunghezze delle sequenze.
- Ogni cella nella TM contiene una direzione tra:
  - **U (up):** movimento verticale.
  - **L (left):** movimento orizzontale.
  - **D (diagonal):** movimento diagonale.

10.5 Procedura di Traceback

Il processo di traceback parte dall'ultima cella della matrice TM ((n, m)) e si muove verso la cella iniziale ($(0, 0)$) per costruire le sequenze allineate. La direzione di ciascun movimento è determinata dal valore presente nella cella della TM:

- **Movimento Verticale (U):** Un gap '-' viene aggiunto alla sequenza *subject*, mentre il nucleotide corrispondente della *query* è posizionato.
- **Movimento Diagonale (D):** I nucleotidi corrispondenti di entrambe le sequenze (*query* e *subject*) vengono aggiunti alle sequenze allineate.
- **Movimento Orizzontale (L):** Un gap '-' viene aggiunto alla sequenza *query*, mentre il nucleotide corrispondente della *subject* è posizionato.

La procedura continua fino a raggiungere la cella $(0, 0)$. A ogni movimento, un valore di punteggio (*score*) è aggiunto al punteggio totale di allineamento (SC) per riflettere la probabilità di similarità tra le sequenze.

10.6 Esempio di Allineamento Utilizzando TM

Consideriamo le seguenti sequenze:

- **Query Sequence:** ACT
- **Subject Sequence:** ACCA

Le sequenze allineate risultanti, generate tramite il processo di traceback, sono:

- **Query:** ACT-
- **Subject:** ACAA

10.7 Riepilogo del Processo

Il processo di traceback permette di:

- Identificare le regioni di similarità tra le sequenze.
- Calcolare il punteggio totale di allineamento (SC).
- Generare sequenze allineate, con gap inseriti ove necessario.

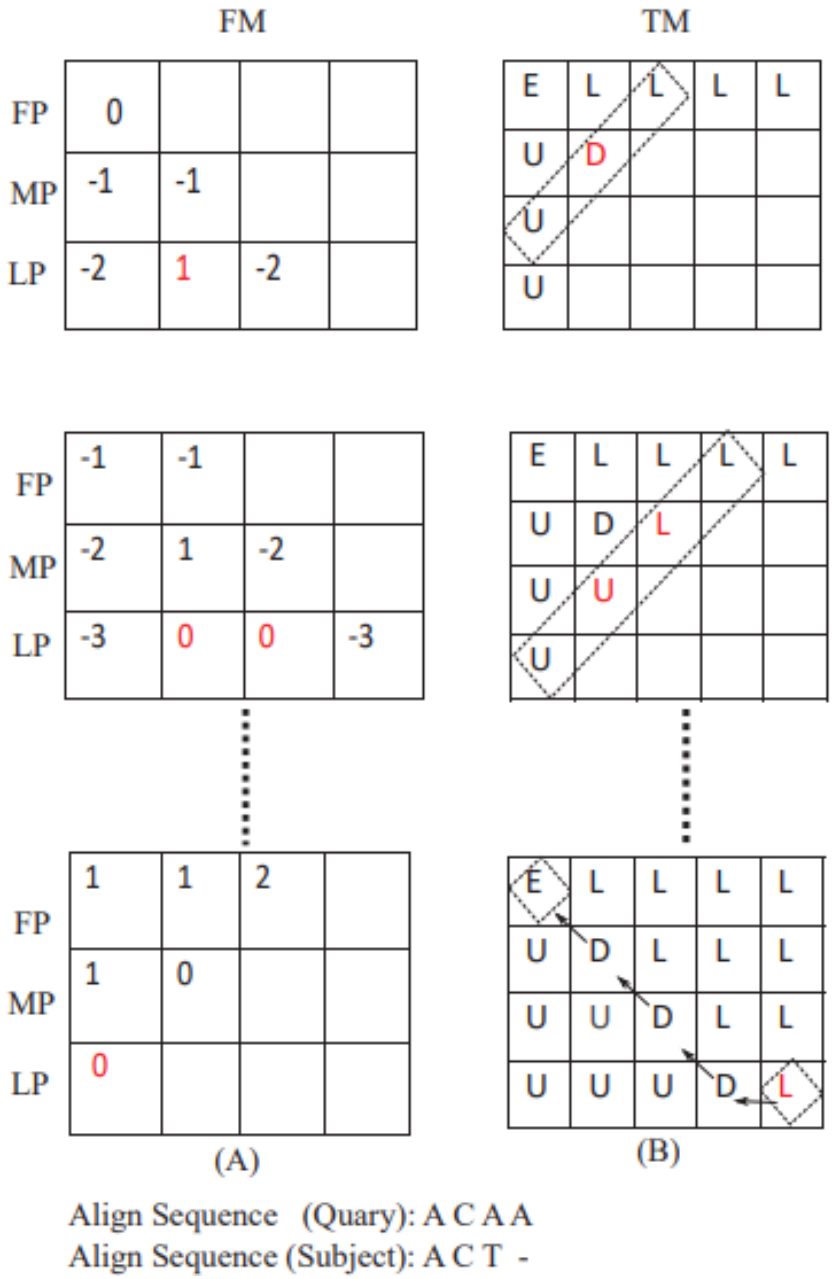

Figura 10.2: Esempio di esecuzione degli algoritmi su due sequenze.

Il risultato finale consente una valutazione accurata della similarità e delle differenze tra le sequenze.

Le matrici iniziali sono mostrate in Figura 10.2. La FM (Figura 10.2 (A)) memorizza i punteggi, mentre la TM (Figura 10.2 (B)) registra i percorsi ottimali. Dopo il completamento, il processo di *traceback* forma l'allineamento ottimale.

Capitolo 11

Analisi delle Performance

L'analisi delle performance del metodo proposto per l'allineamento parallelo di sequenze su GPU dimostra significativi miglioramenti rispetto agli approcci esistenti.

11.1 Complessità Computazionale e Spaziale

Il metodo proposto [5] ottimizza sia la complessità temporale che quella spaziale:

- **Complessità temporale:** Il calcolo della **Foundation Matrix (FM)** è ridotto a $O(n \text{ or } m)$, dove n e m rappresentano le lunghezze delle sequenze. Questo è ottenuto grazie alla generazione parallela dei valori lungo le anti-diagonali, che consente di processare più celle contemporaneamente.
- **Complessità spaziale:**
  - La **Foundation Matrix (FM)** è implementata come una matrice $3 \times ((n \text{ or } m) + 1)$, riducendo lo spazio richiesto a $O(3n)$ or $O(3m)$ rispetto ai tradizionali approcci $n \times m$, che richiedono $O(nm)$.
  - La **Traceback Matrix (TM)** utilizza un formato a 2 bit per cella per memorizzare le direzioni (*up*, *left*, *diagonal*), riducendo ulteriormente lo spazio necessario a $O(n)$ or $O(m)$ per una sequenza.

11.2 Risultati delle Performance

Tabella I: Confronto dei tempi di esecuzione. Il metodo proposto riduce significativamente i tempi di esecuzione. Ad esempio, per una lunghezza di sequenza di 256, il metodo proposto impiega 710.73 µs, a fronte dei 33,541 µs richiesti dal metodo [11].

Tabella 11.1: Time Comparison (in µs) over Different Alignment Algorithms

Sequence Length	[2]	[11]	[9]	[12]	Proposed
16	190.63	274.05	190.21	50.56	48.929
32	330.88	690.41	363.62	100.64	97.28
64	914.38	2375.9	1075.5	192.13	183.97
128	2755	8694.2	3434.8	375.14	356.48
256	9990	33541	12675	791.88	710.73

Tabella II: Confronto dei tempi di trasferimento. Il metodo proposto consuma meno tempo per il trasferimento dati tra host e dispositivo rispetto agli approcci precedenti. Ad esempio, per una sequenza di lunghezza 256, il metodo proposto consuma 292.75 µs per il trasferimento host-to-device.

Tabella 11.2: Data Transfer Time (in µs) Between Device and Host

Sequence Length	Host to Device ([12])	Host to Device (Proposed)	Device to Host ([12])	Device to Host (Proposed)
16	5.69	4.288	3.392	1.632
32	7.712	6.088	3.776	1.856
64	13.024	8.064	3.728	3.192
128	84.545	43.585	70.145	34.44
256	335.52	158.21	297.25	149.28

Di seguito viene riportato un confronto delle performance dei seguenti algoritmi, basato sulle implementazioni discusse nei riferimenti:

- **Needleman-Wunsch (1970) [2]**
  - **Complessità temporale:** $O(n \cdot m)$
  - **Complessità spaziale:** $O(n \cdot m)$
  - Metodo di allineamento globale basato su programmazione dinamica.
  - Utilizzato per l'allineamento completo di due sequenze di lunghezza simile.
- **Matrice di Puntamento - Ray et al. (2016) [9]**
  - **Complessità temporale:** $O(n \cdot m)$
  - **Complessità spaziale:** Ridotta rispetto a Needleman-Wunsch mediante uso di matrici di puntamento.
  - Approccio ottimizzato che riduce l'occupazione di memoria utilizzando una matrice di tracciamento compatta.
  - Adatto per database di grandi dimensioni con elevate esigenze di allineamento.
- **High-Throughput Hardware-Based Alignment - Ray et al. (2016) [11]**
  - **Complessità temporale:** Significativamente ridotta rispetto ai metodi classici grazie all'uso di hardware specializzato.

  - **Complessità spaziale:** Dipendente dall'implementazione hardware (FPGA o GPU).
  - Ideale per applicazioni in tempo reale e per analisi su dataset di grandi dimensioni.

- **GPU-Based Needleman-Wunsch - Chaudhary et al. (2015) [12]**
  - **Complessità temporale:** $O(n \cdot m)$, ma con accelerazione parallela.
  - **Complessità spaziale:** Paragonabile a Needleman-Wunsch, ma gestita su GPU.
  - Utilizza una trasformazione skewing per migliorare la distribuzione del calcolo.
  - Ottimizzato per architetture parallele, offre miglioramenti in velocità rispetto alla versione tradizionale.

11.3 Throughput

La Figura 11.1 mostra il throughput, misurato in **Kilo Sequences Per Second (KSPS)**, per il metodo proposto e gli approcci esistenti ([2], [11], [9], [12]). Il metodo proposto mantiene un throughput superiore e stabile per lunghezze di sequenze crescenti, superando le performance dei metodi confrontati.

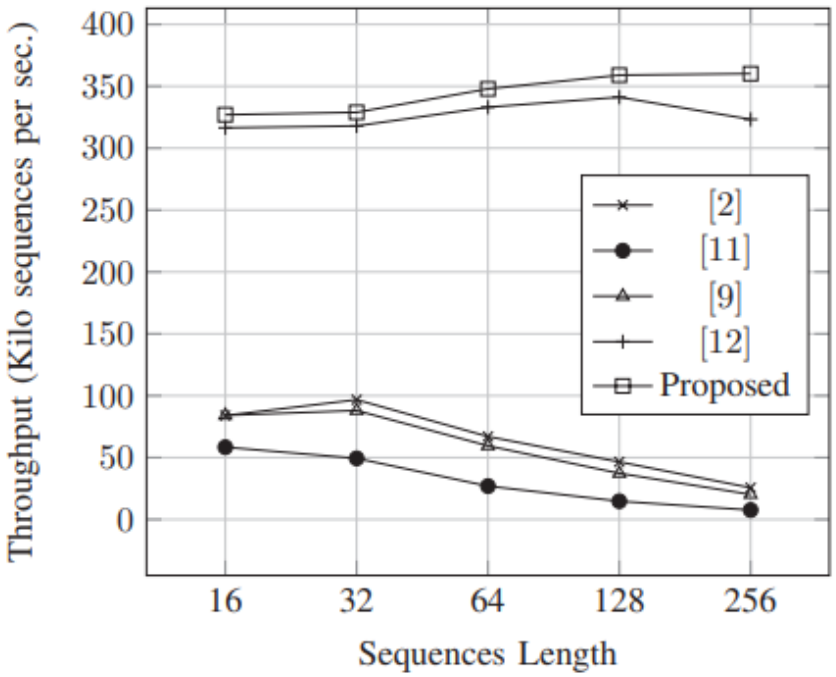

Figura 11.1: Analisi del throughput per lunghezze di sequenze variabili.

11.4 Conclusione Generale

In sintesi, il metodo proposto per l'allineamento parallelo di sequenze biologiche su GPU presenta i seguenti vantaggi:

- Riduzione significativa dei tempi di esecuzione e trasferimento dati.
- Ottimizzazione dello spazio, con una **Foundation Matrix** di $O(3n)$ e una **Traceback Matrix** di $O(n)$ grazie all'uso di rappresentazioni compatte.
- Throughput elevato e stabile, rendendo il metodo particolarmente adatto per sequenze di grandi dimensioni.

Questi miglioramenti fanno del metodo proposto una soluzione efficace e scalabile per l'allineamento di sequenze biologiche.

Capitolo 12

Utilizzo delle IPU negli Algoritmi Bioinformatici e Approcci Moderni

Le Graphcore Intelligence Processing Unit (IPU) rappresentano un'importante evoluzione tecnologica per affrontare problemi computazionali irregolari, come l'allineamento di sequenze biologiche. La loro architettura, basata su memoria SRAM e il modello di calcolo Multiple Instruction Multiple Data (MIMD), permette di migliorare l'efficienza computazionale in applicazioni bioinformatiche ad alta intensità di dati.

12.1 Caratteristiche delle IPU e Ottimizzazioni

Le IPU sono processori paralleli massivamente distribuiti, costituiti da *tiles*, ciascuno dotato di un core e memoria SRAM. L'eliminazione della gerarchia tradizionale della cache consente un accesso rapido ai dati, riducendo i colli di bottiglia tipici delle GPU e delle CPU.
L'algoritmo *X-Drop*, adattato per l'esecuzione su IPU, beneficia di ottimizzazioni specifiche:

- **Riduzione della memoria**: Solo due antidiagonali della matrice di punteggio sono mantenute, riducendo l'impronta di memoria fino a 55 volte rispetto agli approcci classici.
- **Partizionamento basato su grafi**: L'insieme delle sequenze da confrontare è modellato come un grafo, ottimizzando il riutilizzo dei dati e minimizzando il trasferimento tra host e dispositivo.
- **Bilanciamento del carico**: L'uso di sistemi di batching e *work stealing* garantisce un utilizzo uniforme delle risorse computazionali.

12.2 Confronto tra GPU e IPU

Mentre le GPU utilizzano un'architettura SIMD (Single Instruction Multiple Data), ottimizzata per calcoli regolari e lineari, le IPU adottano un modello MIMD (Multiple Instruction Multiple

Data) che consente una gestione più efficiente di accessi irregolari alla memoria. Questo è particolarmente utile per applicazioni bioinformatiche, dove i dati possono essere distribuiti in modo non uniforme.
Le principali differenze sono:

- **Efficienza nei calcoli irregolari**: Le IPU gestiscono meglio i problemi con accessi irregolari ai dati grazie alla memoria SRAM integrata.
- **Riduzione della latenza**: Le IPU riducono la dipendenza dalla memoria esterna, abbassando i tempi di accesso ai dati.
- **Parallelismo fine-grained**: L'architettura MIMD delle IPU permette una maggiore flessibilità nella distribuzione dei task rispetto all'approccio SIMD delle GPU.

12.3 L'Algoritmo X-Drop

L'algoritmo *X-Drop* rappresenta un approccio euristico avanzato per ridurre lo spazio di ricerca durante l'allineamento semi-globale di sequenze biologiche. La sua caratteristica distintiva è l'uso di una condizione dinamica basata sul punteggio corrente, che permette di interrompere il calcolo in regioni non promettenti.

Formula Ricorsiva

La formula ricorsiva di *X-Drop* è definita come:

$$S(i,j) = \max \begin{cases} S(i-1,j-1) + \text{Sim}(H[i], V[j]) & \text{(diagonale)} \\ S(i-1,j) + \text{gap} & \text{(sopra)} \\ S(i,j-1) + \text{gap} & \text{(sinistra)} \end{cases} \tag{12.1}$$

Dove $S(i,j)$ rappresenta il punteggio massimo ottenibile allineando i primi i caratteri di H con i primi j caratteri di V, *Sim* è una funzione di similarità che assegna punteggi positivi ai match e negativi ai mismatch, e *gap* è una penalità per l'inserimento di gap.

Vantaggi dell'Algoritmo

- **Efficienza computazionale**: Riduce il calcolo limitando l'analisi a regioni promettenti della matrice di punteggio.
- **Adattabilità**: Ideale per sequenze lunghe e rumorose, come quelle generate dalle tecnologie di sequenziamento long-read.
- **Riduzione della memoria**: Utilizza una banda dinamica centrata sulle regioni di massimo interesse, evitando il calcolo su tutta la matrice.

Integrazione con Acceleratori Hardware

L'algoritmo *X-Drop* si integra perfettamente con architetture parallele come le IPU grazie a :

- **Calcolo su antidiagonali**: Le IPU memorizzano solo due antidiagonali alla volta, riducendo significativamente l'uso di memoria.
- **Partizionamento dei dati**: Le sequenze sono organizzate in batch ottimizzati per minimizzare i trasferimenti host-dispositivo.
- **Scalabilità**: L'architettura MIMD consente di distribuire i calcoli su molteplici tiles, migliorando il throughput.

12.4 Gli Algoritmi nella Pipeline: ELBA e PASTIS

ELBA (Efficient Long-read Bioinformatic Assembler) è una pipeline per l'assemblaggio de novo di genomi. Essa sfrutta il modello *Overlap-Layout-Consensus* per:

- Effettuare il **conteggio dei k-mer**, individuando substrings comuni tra le sequenze per ridurre il numero di confronti.
- Identificare **regioni di sovrapposizione** tra sequenze tramite l'algoritmo *X-Drop*, migliorando la qualità dei match e riducendo i falsi positivi.
- Assemblare **contig** tramite la semplificazione del grafo di assemblaggio.

Integrando l'algoritmo *X-Drop* sulle IPU, ELBA ottiene un miglioramento delle prestazioni nella fase di sovrapposizione, riducendo i tempi di elaborazione e aumentando l'accuratezza.

PASTIS (Protein Alignment and Similarity Search) è una pipeline progettata per la ricerca di similarità proteica. Essa utilizza *X-Drop* per:

- Estendere e affinare **seed iniziali** basati su matrici di sostituzione (es. BLOSUM62).
- Calcolare **allineamenti semi-globali** per generare matrici di similarità tra sequenze proteiche.

Grazie alla scalabilità delle IPU, PASTIS ottiene significativi vantaggi nella gestione di dataset di grandi dimensioni, migliorando throughput e accuratezza.

Capitolo 13

Implementazione su Hardware Parallelo

L'implementazione dell'algoritmo **X-Drop** su piattaforme parallele come le **Graphcore Intelligence Processing Unit (IPU)** rappresenta un significativo passo avanti rispetto alle implementazioni su CPU e GPU. Le ottimizzazioni sfruttano appieno l'architettura MIMD e la memoria SRAM integrata, migliorando prestazioni e scalabilità.

13.1 Ottimizzazioni per il Calcolo Parallelo

L'algoritmo X-Drop è stato adattato per migliorare l'efficienza computazionale e la scalabilità tramite diverse ottimizzazioni:

- **Antidiagonali Dinamiche**: L'algoritmo memorizza solo due antidiagonali per iterazione, sfruttando variabili temporanee per calcolare i punteggi successivi. Questo approccio riduce il carico di memoria di un fattore $3\times$, rendendolo ideale per l'esecuzione su hardware con memoria limitata come le IPU.
- **Partizionamento Grafico**: Le sequenze vengono organizzate in batch ottimizzati utilizzando un algoritmo di partizionamento grafico, riducendo i trasferimenti host-device e riutilizzando dati comuni.
- **Prefetching e Memorizzazione Locale**: Le IPU utilizzano prefetching per caricare i dati richiesti dalla memoria host prima del calcolo, riducendo i colli di bottiglia associati alla comunicazione.

13.2 Bilanciamento del Carico e Scalabilità

Per massimizzare l'efficienza delle risorse hardware, sono state adottate strategie avanzate:

- **Distribuzione Equa del Lavoro**: Sequenze distribuite tra le unità computazionali in base alla lunghezza e complessità stimata.

- **Work Stealing**: Thread inattivi possono ”rubare” unità di lavoro da quelli attivi, garantendo un utilizzo uniforme.
- **Scalabilità Multi-IPU**: I batch di dati vengono distribuiti su più dispositivi IPU, mantenendo alta l'efficienza anche su dataset complessi come *E. coli* e *C. elegans*.

13.3 Gestione Efficiente della Memoria

Per supportare dataset di grandi dimensioni:

- **Uso della Memoria SRAM**: Informazioni temporanee immagazzinate nella memoria locale delle IPU, eliminando accessi continui alla memoria host.
- **Compressione dei Dati**: I dati intermedi vengono compressi durante il calcolo e decompressi quando necessario, riducendo il consumo di memoria.

13.4 Vantaggi dell'Implementazione su IPU

L'implementazione su IPU offre:

- Riduzione dei tempi di esecuzione fino a $10\times$ rispetto alle GPU per dataset di grandi dimensioni.
- Scalabilità superiore su configurazioni multi-device.
- Minore consumo di memoria grazie a un'allocazione efficiente e al riutilizzo dei dati.

Capitolo 14

Confronto delle Performance

Le implementazioni dell'algoritmo **X-Drop** [1] su IPU dimostrano notevoli vantaggi prestazionali rispetto alle soluzioni tradizionali basate su CPU e GPU. I risultati sperimentali riportati evidenziano i seguenti miglioramenti principali:

14.1 Risultati per Dataset Complessi

Dataset simulati e reali (E. coli e C. elegans):

- Le IPU offrono un'accelerazione fino a **10 volte** rispetto alle GPU, grazie a una gestione più efficiente della memoria e all'accesso ottimizzato ai dati.
- Rispetto alle CPU, le IPU mostrano una velocità di calcolo superiore di **4.65 volte** su dataset complessi come quelli di *C. elegans*, con lunghezze medie di sequenze superiori a 15 kb.

Tabella 14.1: Confronto delle performance tra IPU, GPU e CPU per diversi dataset.

Dataset	Velocità su IPU (GCUPS)	Velocità su GPU (GCUPS)	Velocità su CPU (GCUPS)	Incremento IPU vs GPU (x)	Incremento IPU vs CPU (x)
Simulato	102844	9761.00	50084	10.54	2.05
E. coli 100x	21935	7302.00	1452	3.00	15.12
C. elegans	28587	14860.00	1452	1.92	19.69
PASTIS	16828	—	8449	—	1.99

14.2 Scalabilità e Efficienza

Strong Scaling:

- Per dataset come *E. coli 100x*, l'implementazione su IPU scala quasi linearmente fino a **32 dispositivi**, dimostrando un'efficienza elevata anche con un numero crescente di unità computazionali.
- La tecnica di partizionamento grafico ha ridotto i batch necessari del **52%**, ottimizzando il riutilizzo delle sequenze tra diversi calcoli.

Dataset Proteici (PASTIS):

- Per database di proteine, l'uso delle IPU ha ridotto il tempo di analisi fino a **4.7 volte** rispetto alle CPU, migliorando il throughput generale senza compromettere l'accuratezza.

14.3 Vantaggi Specifici

- **Efficienza nella Memoria:** La riduzione dell'impronta di memoria tramite l'uso di antidiagonali dinamiche e tecniche di compressione consente alle IPU di eseguire analisi su sequenze lunghe senza la necessità di risorse esterne.
- **Utilizzo delle Risorse:** Strategie di bilanciamento del carico come il *work stealing* hanno garantito un utilizzo uniforme dei core computazionali, massimizzando l'efficienza.
- **Tempi di Trasferimento Ottimizzati:** Il prefetching e il partizionamento grafico hanno ridotto drasticamente i tempi di trasferimento tra host e dispositivo, rendendo le IPU una soluzione ideale per dataset di grandi dimensioni.

Capitolo 15

Conclusioni

L'evoluzione degli algoritmi per l'allineamento di sequenze biologiche ha rivoluzionato l'ambito della bioinformatica, rispondendo alle crescenti sfide poste dall'analisi di dati biologici di grandi dimensioni. Gli approcci classici, come Needleman-Wunsch e Smith-Waterman, rimangono fondamentali per la loro capacità di produrre risultati accurati attraverso tecniche di programmazione dinamica. Tuttavia, l'aumento esponenziale dei dati biologici ha reso necessaria l'introduzione di tecniche più efficienti, sia in termini computazionali che di gestione della memoria.
L'introduzione delle Graphcore IPU e l'adattamento di algoritmi come X-Drop rappresentano una svolta significativa. Grazie alla loro architettura innovativa, le IPU consentono di sfruttare tecniche avanzate di parallelizzazione e gestione della memoria, superando i limiti delle GPU e delle CPU nei calcoli irregolari. Il lavoro svolto con le pipeline ELBA e PASTIS ha dimostrato il potenziale di queste soluzioni per l'assemblaggio genomico e la ricerca di omologia proteica, offrendo accelerazioni fino a 10 volte rispetto alle GPU e 4,65 volte rispetto alle CPU.

15.1 Principali Contributi

- **Ottimizzazioni Tecniche:** Attraverso tecniche come il partizionamento grafico, il prefetching e l'utilizzo di antidiagonali dinamiche, le IPU hanno mostrato un'efficienza computazionale superiore.
- **Prestazioni Scalabili:** I risultati hanno evidenziato una scalabilità quasi lineare, con efficienze elevate anche su configurazioni multi-IPU.
- **Applicazioni Pratiche:** Le pipeline testate, integrate con X-Drop, hanno permesso analisi rapide e precise su dataset complessi come quelli di *E. coli* e *C. elegans*.

15.2 Prospettive Future

L'implementazione su IPU rappresenta solo un passo verso l'evoluzione dell'analisi bioinformatica. I futuri miglioramenti tecnologici potrebbero includere:

- Maggiore integrazione con tecnologie di intelligenza artificiale.
- Ottimizzazione delle comunicazioni tra host e dispositivo per ulteriori riduzioni dei tempi di trasferimento.
- Sviluppo di algoritmi ancora più adattabili alle architetture parallele.

In conclusione, l'utilizzo delle IPU negli algoritmi di allineamento apre nuove opportunità per la bioinformatica computazionale, stabilendo un nuovo standard per l'analisi di sequenze biologiche su larga scala. Tali innovazioni rappresentano il punto di incontro tra la biologia e la potenza computazionale moderna, permettendo di affrontare le sfide future con strumenti più avanzati ed efficienti.

Bibliografia

[1] L. Burchard, M.X. Zhao, J. Langguth, A. Buluc, and G. Guidi. Space efficient sequence alignment for sram-based computing: X-drop on the graphcore ipu. 2023.

[2] Mark Halsey. *Introduction to Computational Molecular Biology: Divide and Conquer: More Efficient Dynamic Programming*. 2004.

[3] Zsuzsanna Lipták. Pairwise alignment. 2018.

[4] S.B. Needleman and C.D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[5] W.R. Pearson and D.J. Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences*, 85(8):2444–2448, 1988.

[6] A. Sarkar, K. Ray, D. Chowdhury, K. Sahu, S. Kundu, and S. Ghosh. Time and space efficient optimal pairwise sequence alignment using gpu. In *2019 IEEE Region 10 Conference (TENCON 2019)*, 2019.

[7] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.