Analisi dell'Algoritmo per la Ricerca di Intervalli nel Suffix Array Utilizzando Burrows-Wheeler Alignment Tool: Efficienza e Applicazioni per il Pattern Matching di Short Read

Irene Gaita
Mat: 0522501839
i.gaita@studenti.unisa.it

ABSTRACT

Il rapido sviluppo delle tecnologie di sequenziamento del DNA, in particolare il Next Generation Sequencing (NGS), ha reso fondamentale l'allineamento delle letture brevi (short read) su un genoma di riferimento per estrarre informazioni biologiche significative. Questo lavoro presenta un'analisi approfondita di un algoritmo chiave proposto nella letteratura scientifica, incentrato sull'ottimizzazione della ricerca di intervalli nel Suffix Array tramite la trasformata di Burrows-Wheeler (BWT) e l'FM-index, implementato nel Burrows-Wheeler Alignment Tool (BWA). Lo studio in analisi è stato sviluppado da Li et al. "Fast and accurate short read alignment with Burrows–Wheeler transform" [1].

Attraverso una disamina dettagliata delle tecniche di hashing, strutture dati come gli alberi dei suffissi e l'integrazione di approcci basati su machine learning, il lavoro presentato esplora i progressi sull'argomento e in particolare l'algoritmo BWA rispetto ad altre metodologie. I risultati sperimentali confermano i vantaggi di BWA in termini di velocità, efficienza della memoria e accuratezza, evidenziando come l'algoritmo affronti efficacemente le sfide dell'allineamento moderno.

Infine, viene discusso il potenziale miglioramento dell'algoritmo tramite nuove strategie, come la frammentazione delle letture, per affrontare scenari complessi con letture lunghe o elevata variabilità genomica. Questa analisi sottolinea l'importanza di strumenti come BWA nel panorama bioinformatico, ma contribuisce a delineare opportunità per la gestione dei big data genomici.

1 INTRODUZIONE

Negli ultimi anni, i progressi nelle tecnologie di sequenziamento del DNA hanno rivoluzionato la genomica, permettendo di acquisire enormi quantità di dati genetici in modo rapido ed economico. Tra queste tecnologie, il sequenziamento di nuova generazione (Next Generation Sequencing, NGS) si è affermato come uno degli strumenti principali per esplorare e comprendere la complessità del genoma di diversi organismi. Le piattaforme NGS, come quelle sviluppate da Illumina/Solexa, sono in grado di generare milioni di sequenze di DNA (short read) in una singola esecuzione. Ogni short read rappresenta un breve frammento di una sequenza di DNA più lunga, che viene successivamente analizzato e ricostruito per dedurre informazioni biologiche significative.

Tuttavia, la generazione di questi dati su larga scala porta con sé sfide notevoli. Uno dei problemi principali riguarda la fase di pattern matching delle letture (short read alignment), ovvero l'operazione mediante la quale i frammenti di DNA devono essere mappati su un genoma di riferimento. Questo processo è essenziale per estrarre informazioni rilevanti dai dati grezzi, consentendo di interpretare correttamente la sequenza genomica. Le letture, infatti, devono essere confrontate con una sequenza di riferimento per determinarne la posizione corretta all'interno del genoma. Tuttavia, l'orientamento e la posizione esatta di ciascuna lettura non sono noti a priori, complicando ulteriormente il processo di allineamento.

La sfida principale nel mapping delle letture short read risiede nella necessità di un'elevata accuratezza, mantenendo al contempo una gestione efficiente delle risorse computazionali. Considerato il grande volume di dati generati, è fondamentale che gli algoritmi utilizzati per mappare queste letture siano in grado di bilanciare precisione, velocità di esecuzione e consumo di memoria. Questa operazione diventa particolarmente critica quando si lavora con dati provenienti da organismi complessi, come il genoma umano, che presenta ripetizioni, variabilità e altre caratteristiche strutturali che possono rendere difficile il corretto allineamento delle sequenze.

Le applicazioni per il pattern matching di letture short read sono numerose e variegate. Comprendere come le sequenze biologiche si allineano al genoma è fondamentale per svelare il "progetto genetico" di un organismo vivente, permettendo agli scienziati di decifrare le istruzioni che regolano la vita a livello molecolare. Le informazioni estratte da questi dati possono fornire indicazioni preziose sui processi evolutivi, sulle cause di malattie genetiche, e sull'interazione tra organismi e ambiente. La ricerca bioinformatica si trova oggi di fronte a un cambiamento di paradigma: se in passato la sfida principale era l'acquisizione dei dati di sequenza, oggi l'attenzione si è spostata verso la memorizzazione, l'elaborazione e l'analisi efficace di questi dati.

L'ottimizzazione delle risorse computazionali per gestire questi enormi volumi di dati è diventata cruciale per consentire una ricerca genomica su larga scala. Per questo motivo, sono stati sviluppati diversi programmi specializzati per il mapping, che si possono suddividere in categorie distinte.

Il primo gruppo di programmi, che include RMAP [2], MAQ [3], ZOOM [4], SeqMap [5] e CloudBurst [6], utilizza una tecnica di hashing delle letture per individuare le corrispondenze con la sequenza di riferimento. Questo approccio richiede meno memoria rispetto agli strumenti che indicizzano il genoma completo, poiché si concentra sull'hashing delle letture stesse. Tuttavia, spesso richiede la scansione completa del genoma di riferimento, rendendo l'allineamento più lento quando solo una piccola parte delle letture ha effettivamente bisogno di essere mappata.

Un secondo gruppo di programmi, come SOAPv1 [7], PASS [8], MOM [9], NovoAlign, ReSEQ, Mosaik e BFAST, adotta un approccio differente, eseguendo l'hashing del genoma di riferimento. Questo

metodo crea un indice dell'intero genoma, permettendo di allineare rapidamente le letture al genoma indicizzato senza doverlo scansionare ogni volta. Sebbene questa tecnica migliori la velocità per grandi dataset, richiede quantità significative di memoria per costruire e memorizzare l'indice. Inoltre, la loro strategia iterativa può risentire del tasso di errore del sequenziamento, rallentando il processo di allineamento in presenza di errori elevati.

Infine, una terza categoria di software, rappresentata da strumenti come Slider [10] e SOAPv2 , utilizzano un metodo basato sull'ordinamento (merge-sorting) sia delle sottosequenze del genoma di riferimento che delle letture. Questo approccio si differenzia dai precedenti perché non richiede né l'hashing delle letture né l'hashing del genoma completo, e può risultare utile in particolari applicazioni dove l'ordinamento offre vantaggi specifici nella gestione delle sequenze.

Recentemente, la teoria del pattern matching basato sulla trasformata di Burrows-Wheeler (BWT) ha attirato l'attenzione di diversi gruppi di ricerca, portando allo sviluppo di software come Bowtie e BWA[11].

Il lavoro di Li et al. [1] ha introdotto l'algoritmo BWA (Burrows-Wheeler Aligner), dimostrando che l'uso della trasformata di Burrows-Wheeler (BWT) consente un allineamento rapido e preciso, riducendo significativamente i tempi di calcolo e migliorando la sensibilità rispetto agli algoritmi precedenti.

Successivamente, l'elaborato di S. Kumar et al. [11] ha proposto un approccio combinato che integra la BWT con le strutture ad albero wavelet. I risultati hanno mostrato un miglioramento

dell'efficienza nell'allineamento, con una riduzione dei tempi di accesso ai dati, rendendo l'intero processo più veloce e scalabile.

Un contributo importante è stato fornito da Kelly M. Robinson et al. [12], il quale hanno dimostrato che l'ottimizzazione degli allineatori specificamente progettata per dati multi-specie può aumentare significativamente l'accuratezza e ridurre i tempi di calcolo. Questo studio ha sottolineato l'importanza di affinare gli algoritmi per diverse tipologie di dati, ottenendo risultati superiori in scenari complessi.

In un'ottica di innovazione, Y. Jung et al. [13] hanno esplorato l'applicazione del machine learning all'emulazione dell'algoritmo BWA-MEM. I risultati hanno indicato che l'approccio predittivo ha portato a un'ulteriore riduzione dei tempi di allineamento, suggerendo che l'integrazione di tecnologie emergenti come

l'intelligenza artificiale può migliorare le prestazioni degli strumenti di bioinformatica.

Inoltre, C. Kim et al.[14] hanno dimostrato come sia possibile accelerare il mapping delle sequenze genomiche su server comuni, evidenziando che infrastrutture di calcolo meno costose possono gestire efficacemente carichi di lavoro complessi, rendendo l'analisi genomica più accessibile a un pubblico più ampio.

Infine, H.Li [15] ha presentato un'analisi dettagliata delle applicazioni pratiche di BWA-MEM, evidenziando la sua versatilità nell'allineamento di letture sequenziali, sequenze di cloni e contig di assemblaggio. I risultati hanno mostrato che BWA-MEM è capace di affrontare diverse tipologie di dati e scenari sperimentali, confermando la sua utilità in vari contesti di ricerca.

Questi studi forniscono un quadro chiaro delle soluzioni emergenti per affrontare le sfide dell'allineamento delle letture short read, sottolineando i progressi compiuti nella bioinformatica e aprendo la strada a ulteriori sviluppi.

In questo contesto di innovazione, il seguente studio si propone di approfondire ulteriormente questi temi, concentrandosi sull'importanza dell'algoritmo per la ricerca di intervalli SA applicato al tool della Burrowa-whieeler Alignmwnt per l'allineamento di brevi sequenze di lettura.

2 RELATED WORKS

In questo studio ci concentriamo sul lavoro di Li et al. [1] sull'impiego dell'algoritmo Burrows-Wheeler Aligner (BWA) insieme alla Burrows-Wheeler Transform (BWT) insieme per migliorare le operazioni di allineamento e confronto tra genomi.

Tradizionalmente, per affrontare il problema dell'allineamento e della rilevazione delle varianti, sono stati utilizzati approcci come la ricerca esaustiva (brute force) e strutture dati come i trie.

L'approccio brute force, però, si rivela poco pratico per confrontare direttamente il genoma di riferimento con quelli individuali, poiché esplorare tutte le possibili corrispondenze richiede risorse computazionali elevate. Anche l'uso dei suffix trie, sebbene fornisca una struttura ad albero efficiente per il confronto delle stringhe, diventa oneroso in termini di memoria quando si applica a sequenze genomiche estese.

La combinazione di BWT e BWA offre una soluzione più efficiente: essa riduce il costo computazionale degli allineamenti grazie a una rappresentazione compressa del genoma. Questo metodo permette di assemblare i genomi individuali in riferimento al genoma umano di base, facilitando l'identificazione delle varianti in modo più rapido e preciso.

2.1 Brute Force per il Pattern Matching

L'approccio brute force per la ricerca di pattern in una sequenza genomica consiste nello scorrere sequenzialmente il pattern lungo l'intero genoma alla ricerca di corrispondenze esatte. In pratica, si esamina ogni posizione possibile della sequenza genomica e si verifica, carattere per carattere, se il pattern corrisponde esattamente alla sottosequenza in quella posizione. Questo processo richiede di ripetere il confronto per ciascuna posizione nel genoma, spostandosi successivamente di un carattere e confrontando nuovamente il pattern con la nuova sottosequenza generata.

Dal punto di vista computazionale, l'approccio brute force risulta altamente inefficiente per sequenze lunghe, poiché implica un elevato numero di confronti. Se consideriamo un genoma di lunghezza m e un pattern di lunghezza n, la complessità temporale

dell'algoritmo è $O(m \cdot n)$ nel caso peggiore. Questo è dovuto al fatto che l'algoritmo deve scorrere tutte le m posizioni del genoma e, per ciascuna di queste, confrontare ogni carattere del pattern, generando un numero di operazioni pari a $m \times n$ complessivamente.

Questa complessità rende il metodo impraticabile per genomi di grandi dimensioni, come il genoma umano, che conta circa 3 miliardi di basi azotate. Di conseguenza, il tempo richiesto cresce linearmente con la lunghezza del genoma e del pattern, con un impatto significativo sui requisiti computazionali e sulla durata dell'analisi [16].

2.2 Suffix tree

L'approccio suffix tree, consente di comprimere l'informazione di tutti i pattern da cercare in un'unica struttura ad albero, evitando di dover ripetere il confronto lungo l'intero genoma per ogni pattern. In un suffix tree, ogni nodo rappresenta un carattere e ogni percorso dalla radice a una foglia rappresenta una sottosequenza (o pattern) del genoma.

La costruzione di un suffix trie richiede che tutti i suffissi della sequenza siano inseriti nell'albero, con ogni suffisso rappresentato come un cammino dalla radice alla foglia. Questo processo ha una complessità di $O(n)$, dove n è la lunghezza del genoma, poiché ciascun suffisso viene inserito individualmente.

Una volta costruito il suffix trie, il pattern matching può essere eseguito in modo rapido: la ricerca di un pattern richiede solo il percorso lungo i nodi della struttura, il che risulta essere pari a $O(m)$ con m pari alla lunghezza del pattern. Il vantaggio principale è che, avendo memorizzato tutti i pattern all'interno dell'albero, è possibile eseguire il matching simultaneamente senza ripetere i confronti.

Un problema importante del suffix tree, tuttavia, è l'assenza di informazioni sulla posizione esatta delle corrispondenze all'interno del genoma. Per risolvere questo, si aggiunge il simbolo speciale "$" in ogni foglia in modo da indicare la posizione iniziale del suffisso che porta a quella foglia. In questo modo, l'informazione di posizione è preservata e l'albero è compresso, rendendo possibile identificare rapidamente dove avviene la corrispondenza all'interno del genoma [17]. La Figura 1 rappresenta un esempio del suffix tree dove i sei percorsi dalla radice alla foglia (mostrati come riquadri) corrispondono ai sei suffissi A$, NA$, ANA$, NANA$, ANANA$ e BANANA$. I numeri nelle foglie danno la posizione iniziale del suffisso corrispondente. I collegamenti suffisso, disegnati tratteggiati, vengono utilizzati durante la costruzione.

Un problema significativo nell'approccio suffix tree è la necessità di costruire completamente l'albero prima di poter applicare una compressione, il che comporta un elevato consumo di memoria lungo il percorso di costruzione. Nonostante la notazione $O(n)$ indichi una complessità teorica lineare rispetto alla lunghezza del genoma n, essa ignora una costante importante: la memoria effettiva necessaria per implementare un suffix tree può essere fino a 20 volte maggiore della lunghezza del genoma. Questo elevato consumo di memoria è dovuto anche alle numerose ripetizioni presenti nelle sequenze genomiche, che incrementano ulteriormente le dimensioni della struttura.

Per affrontare questi limiti di memoria, sono stati sviluppati algoritmi più efficienti in termini di uso della memoria, che permettono la costruzione del suffix tree direttamente in tempo reale (*runtime*), ottimizzando così l'impiego di risorse senza compromettere la complessità lineare rispetto alla lunghezza del genoma [18]. Tuttavia, l'approccio basato sulla trasformata di Burrows-Wheeler (*BWT*) rappresenta un'alternativa ancora più efficiente e compatta, poiché codifica il genoma in una struttura compressa che riduce drasticamente il consumo di memoria mantenendo l'accuratezza nell'allineamento.

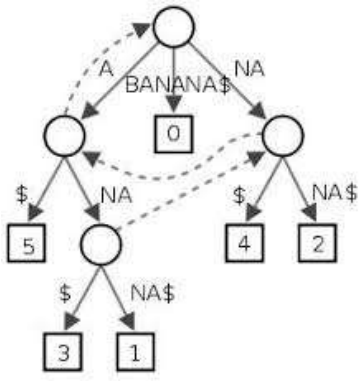

Figure 1: Esempio della costruzione del suffix tree per la stringa BANANA.

2.3 Burrows-Wheeler Transform (BWT)

Sia dato un alfabeto Σ, e il simbolo speciale $ che non appartiene a Σ ed è considerato lessicograficamente più piccolo di tutti i simboli in Σ. Una stringa $X = a_0, a_1, \ldots, a_{n-1}$ termina sempre con il simbolo $ (cioè, $a_{n-1} =$ $) e questo simbolo appare solo alla fine della stringa. La notazione $X[i] = a_i$ indica il simbolo alla posizione i-esima di X, dove i varia da 0 a $n-1$. La sottostringa $X[i, j]$ rappresenta la sequenza di simboli da a_i a a_j, mentre $X_i = X[i, n-1]$ denota il suffisso che inizia alla posizione i della stringa X.

Una **rotazione ciclica** di una stringa S è una stringa della forma $y \cdot x$, dove $S = x \cdot y$ e x, y sono sottostringhe di S. L'operatore $\cdot$ indica la concatenazione. Ad esempio, se $S =$ banana$, allora ana$ban è una rotazione ciclica di S (con $y =$ ana$ e $x =$ ban). La matrice delle permutazioni cicliche della stringa S, verrà poi ordinata secondo l'ordine alfabetico delle loro sequenze. La matrice dei caratteri ordinati risultante rappresenterà la matrice BWT.

Definiamo quindi BWT (S) come la stringa formata dai caratteri presenti nell'ultima colonna della matrice.

La Figura 2 mostra un esempio di costruzione dell'array dei suffissi e della BWT. L'algoritmo illustrato è di complessità quadratica in termini di tempo e spazio [19].

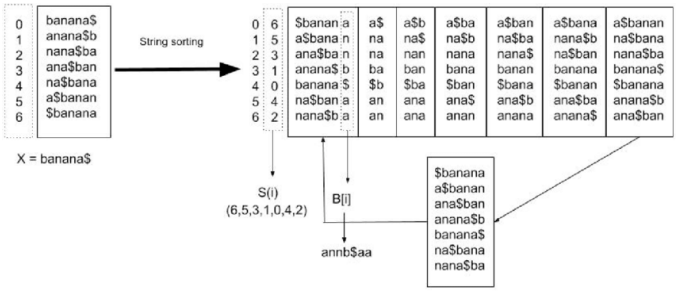

Figure 2: Esempio della costruzione della BWT: La matrice delle rotazioni (Rotation Matrix) e la matrice ordinata (Ordered Matrix) della stringa "BANANA$". L'output della BWT è "ANNB$AA".

2.3.1 Last-to-First Mapping. Per recuperare la stringa originale da una stringa trasformata (BWT), si può utilizzare la "tecnica di inversione BWT". La prima colonna della matrice di rotazioni BWT può essere conosciuta gratuitamente, poiché è semplicemente l'ordinamento lessicografico della stringa trasformata. Ogni carattere dell'ultima colonna ha un corrispondente in questa prima colonna, e sfruttando questa relazione, è possibile risalire alla stringa iniziale.

In particolare, la prima occorrenza di un carattere nell'ultima colonna corrisponde alla sua prima occorrenza nella prima colonna (ordinata in ordine lessicografico). Questo principio è alla base dell'algoritmo LF-mapping (Last-to-First Mapping), che permette di seguire la traccia di un carattere dalla fine della stringa trasformata verso l'inizio, per ricostruire la stringa originale passo per passo. La Figura 3 mostra un esempio di una corrispondenza.

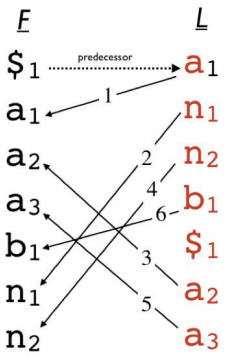

Figure 3: Last-to-First Mapping

Questa compressione è particolarmente vantaggiosa per sequenze con molte ripetizioni locali, come i dati genomici. Tuttavia, anche l'implementazione della BWT può essere costosa in termini di memoria, poiché costruire la matrice completa richiede memoria quadratica rispetto alla lunghezza della stringa originale. Diverse tecniche, come gli algoritmi basati su suffissi, riducono l'uso di memoria evitando la costruzione esplicita dell'intera matrice di rotazioni.

Ferragina e Manzini hanno proposto una variante chiamata FM-index, un indice basato su BWT che consente di effettuare ricerche di pattern in modo efficiente sia in termini di spazio che di tempo[20].

2.4 FM-index

L'FM-index è una struttura dati progettata per l'allineamento efficiente di brevi sequenze di lettura (short reads) a un genoma di riferimento. Si basa sulla Trasformata di Burrows-Wheeler (BWT) e si distingue per la sua efficienza in termini di spazio, occupando una quantità di memoria proporzionale alla compressione ottenuta dalla BWT. Ulteriori ottimizzazioni, come il calcolo on-the-fly di alcuni elementi, contribuiscono a ridurre ulteriormente l'utilizzo di memoria. L'algoritmo utilizza tre tabelle principali:

- **Suffix Array (SA)**: contiene le posizioni originali delle voci ordinate nella matrice BWT. In altre parole, SA[i] indica la posizione nel testo originale del suffisso che occupa la posizione i nella matrice BWT.
- **Occurrence Table (OCC)**: memorizza il numero cumulativo di occorrenze di ciascun simbolo dell'alfabeto in un prefisso della stringa BWT. Ad esempio, OCC[i][A] rappresenta il numero di 'A' presenti nei primi i simboli della stringa BWT. Questa tabella è fondamentale per la ricerca backward.
- **CountTable(C)**:indica il numero cumulativo di simboli lessicograficamente inferiori a un dato simbolo nell'intero testo originale. In pratica, C[b] fornisce la posizione iniziale nella matrice BWT dei suffissi che iniziano con il simbolo b.

Grazie a queste tabelle, l'FM-index permette di:

- Verificare se una stringa di query W è una sottostringa del testo originale X in tempo O(|W |).
- Contare il numero di occorrenze di W in X in tempo O(|W|).
- Identificare le posizioni di tutte le occorrenze di W in X.

L'indice occupa uno spazio O(N) dove N è la lunghezza del testo originale. Ciò significa che la memoria utilizzata cresce lin- earmente con la dimensione del testo, rendendo l'FM-index una struttura dati efficiente per la gestione di genomi di grandi dimen- sioni.

2.5 Backward Search e raffinamento dell'Intervallo nell'FM-index

La ricerca di una stringa di query W all'interno di un testo X utilizzando l'*FM-index* si basa su un processo iterativo che prevede la ricerca iniziale di un intervallo nella matrice BWT e il suo successivo raffinamento. Questo processo sfrutta le proprietà della BWT e le informazioni contenute nelle tabelle C e OCC dell'*FM-index*.

Ricerca Iniziale.

(1) **Inizio dalla fine della query:** La ricerca inizia considerando l'ultimo carattere della stringa di query, $W[|W|-1]$.
(2) **Utilizzo della tabella *(C):*** La tabella C fornisce la posizione iniziale nella matrice BWT dei suffissi che iniziano con il carattere $W[|W|-1]$. L'intervallo iniziale $[k,l]$ viene definito come:

$$k = C[W[|W|-1]] + 1,$$

$l = C[W[|W|-1]+1]$, dove $W[|W|-1]+1$ rappresenta

il carattere successivo a $W[|W|-1]$ nell'ordine lessicografico.

(3) **Intervallo iniziale come punto di partenza:** La ricerca iniziale nell'FM-index è fondamentale perché stabilisce il punto di partenza per l'algoritmo di ricerca. Essa definisce un intervallo iniziale nella matrice BWT che contiene tutte le posizioni candidate in cui la stringa di query W potrebbe iniziare.

Raffinamento dell'Intervallo.

(1) **Iterazione sui caratteri della query:** Il processo di raffinamento procede iterativamente, analizzando i caratteri di W dalla fine all'inizio, da $W[|W|-2]$ fino a $W[0]$.

(2) **Utilizzo della tabella *(OCC)*:** Per ogni carattere $W[i]$, si utilizza la tabella *OCC* per aggiornare l'intervallo $[k, l]$ in modo che includa solo i suffissi che iniziano con la sottostringa $W[i, |W|-1]$. L'aggiornamento avviene in questo modo:

$$k = C[W[i]] + OCC[k-1][W[i]] + 1$$

$$l = C[W[i]] + OCC[l][W[i]]$$

(3) **Stretto intervallo finale:** Ad ogni iterazione, l'intervallo $[k, l]$ si restringe, escludendo i suffissi che non corrispondono alla sottostringa corrente di W.

Esempio. Supponiamo di avere il testo $X =$ "*banana*\$ e la query $W =$ "*ana*". La matrice *BWT* di X è:

annb$aa

La tabella C è:

$$C['\$'] = 0, \quad C['a'] = 1, \quad C['b'] = 4, \quad C['n'] = 5$$

E la tabella *OCC* è:

i	1	2	3	4	5	6	7
OCC['$']	0	0	0	0	1	1	1
OCC['a']	1	1	1	1	1	2	3
OCC['b']	0	0	0	1	1	1	1
OCC['n']	0	1	2	2	2	2	2

Ricerca Iniziale:

- $W[|W|-1] =' a'$
- $k = C['a'] + 1 = 2$
- $l = C['b'] = 4$
- Intervallo iniziale: $[k, l] = [2, 4]$

Raffinamento:

- **Iterazione 1 ($W =' n'$):**

$$k = C['n'] + OCC[k-1]['n'] + 1 = 5 + 0 + 1 = 6$$

$$l = C['n'] + OCC[l]['n'] = 5 + 2 = 7$$

  Nuovo intervallo:$[k, l] = [6, 7]$
- **Iterazione 2 ($W =' a'$):**

$$k = C['a'] + OCC[k-1]['a'] + 1 = 1 + 1 + 1 = 3$$

$$l = C['a'] + OCC[l]['a'] = 1 + 3 = 4$$

  Intervallo finale:$[k, l] = [3, 4]$
  Questo intervallo rappresenta le posizioni nella BWT che terminano con "ana".

Conclusioni. Il processo di ricerca iniziale e raffinamento dell'intervallo nell'*FM-index* permette di individuare in modo efficiente le occorrenze di una stringa di query all'interno di un testo. La chiave di questo processo risiede nell'uso combinato delle tabelle C e *OCC*, che consentono di restringere iterativamente l'intervallo di ricerca fino a identificare le corrispondenze esatte.

2.6 Allineamento inesatto

A differenza dell'allineamento esatto, che richiede una corrispondenza perfetta tra due sequenze, l'allineamento inesatto si propone di trovare allineamenti ottimali anche in presenza di differenze tra le sequenze, come mismatch o gap. Questo tipo di allineamento è particolarmente importante in bioinformatica, dove le sequenze di DNA o RNA possono presentare variazioni dovute a mutazioni, errori di sequenziamento o eventi di splicing.

L'allineamento inesatto basato sulla *Burrows-Wheeler Transform* (BWT) offre un approccio efficiente per affrontare questo problema. Sfruttando la BWT e la *backward search*, è possibile esplorare lo spazio di ricerca delle possibili corrispondenze tra una sequenza di query e una sequenza di riferimento in modo rapido.

Un elemento chiave di questo approccio è l'utilizzo di un array, $D(\cdot)$, che memorizza il limite inferiore del numero di differenze (mismatch o gap) presenti in un prefisso della sequenza di query. Questo array consente di limitare lo spazio di ricerca durante la *backward search*, evitando di esplorare rami dell'albero dei prefissi che non possono portare a un allineamento valido.

3 BURROWS-WHEELER ALIGNMENT TOOL (BWA)

Il *Burrows-Wheeler Alignment tool* (BWA) è un pacchetto software progettato per allineare in modo rapido ed efficiente brevi sequenze di lettura (*reads*) a una sequenza di riferimento di grandi dimensioni, come il genoma umano. Questo strumento supporta la gestione di mancati allineamenti e gap, garantendo flessibilità nell'analisi delle sequenze.

Lo sviluppo di BWA è stato motivato dalla crescente necessità di programmi di allineamento rapidi e precisi, capaci di gestire l'enorme quantità di dati prodotti dalle nuove tecnologie di sequenziamento del DNA. BWA appartiene a una terza generazione di strumenti di allineamento, emersa dopo i metodi basati su tabelle hash delle *reads* e quelli basati su tabelle hash del genoma. Questa nuova generazione include software come SOAPv2 e Bowtie e sfrutta la trasformata di Burrows-Wheeler (BWT) per ottimizzare l'allineamento delle sequenze.

3.1 Perché BWA utilizza l'FM-index e la BWT?

BWA si basa sull'FM-index, una struttura dati costruita a partire dalla BWT, per allineare le sequenze. Questo approccio offre numerosi vantaggi rispetto ai metodi basati su tabelle hash:

- **Efficienza:** Grazie alla ricerca all'indietro (*backward search*) con la BWT, è possibile simulare in modo efficace l'attraversamento dall'alto verso il basso del *trie* dei prefissi del genoma, utilizzando una quantità relativamente ridotta di memoria.
- **Velocità:** La ricerca di corrispondenze esatte con la BWT può essere effettuata in tempo $O(m)$, dove m è la lunghezza della stringa di query, indipendentemente dalla dimensione del genoma.
- **Gestione delle ripetizioni:** La BWT comprime le ripetizioni esatte in un unico percorso nel *trie* dei prefissi, eliminando la necessità di allineare le *reads* a ciascuna copia della ripetizione.

Il software BWA comprende tre algoritmi distinti: BWA-ALN (o BWA-backtrack), BWA-SW e BWA-MEM.

BWA-ALN(o BWA-backtrack). Il primo algoritmo sviluppato, BWA-ALN, è stato progettato per l'allineamento di brevi letture di sequenziamento, tra 36 e 100 bp, supportando gap e mismatch per identificare inserzioni e delezioni. Al momento della sua introduzione, BWA-ALN risultava significativamente più veloce (10-20 volte) rispetto ad altri strumenti disponibili all'epoca, senza comprometterne l'accuratezza.

BWA-SW. In seguito, è stato sviluppato BWA-SW per eseguire il mapping di letture con lunghezze tra 200 bp e 1 Mbp. Con l'aumento della lunghezza delle letture, è cresciuta anche la probabilità di riscontrare errori e variazioni strutturali nelle sequenze, richiedendo un metodo più flessibile. Per questo, BWA-SW adotta un approccio di allineamento locale, che si dimostra più efficace nel gestire tali errori e variazioni. Tuttavia, con l'aumento della mole di dati provenienti dai grandi progetti genomici e la tendenza a ridurre le lunghezze di lettura in molte applicazioni, BWA-SW ha iniziato a rivelare alcune limitazioni nel trattare volumi così elevati di dati.

BWA-MEM. L'algoritmo più recente, BWA-MEM, è stato pensato per l'allineamento rapido e accurato di letture tra 100 e 1 Mbp rispetto a un genoma di riferimento di grandi dimensioni. BWA-MEM supporta allineamenti chimerici, è in grado di gestire errori di sequenziamento e può scegliere tra allineamento locale o globale, rendendolo ideale per letture superiori a 70 bp e fino a 1 Mbp.

BWA-MEM adotta un approccio *seed-and-extend*, in cui i pattern vengono individuati inizialmente come "seed" e poi estesi ai lati per formare un allineamento completo. Per aumentare la precisione, BWA-MEM estende questi pattern utilizzando un algoritmo *affine-gap* basato su Smith-Waterman [21], che consente di gestire in modo flessibile inserzioni e delezioni grazie a penalità adattative per i gap.

4 ANALISI DELL'ALGORITMO PER LA RICERCA DI INTERVALLI NEL SUFFIX ARRAY

Il presente lavoro analizza l'algoritmo descritto dallo studio di Li et al. [1] che illustra una procedura ricorsiva per la ricerca degli intervalli inSA (*Suffix Array*) delle sottostringhe di X che corrispondono alla stringa di query W con non più di z differenze.

La procedura `CalculateD` fornisce un metodo per calcolare in modo efficiente i valori dell'array $D(\cdot)$. Utilizzando la BWT della sequenza di riferimento inversa, è possibile verificare rapidamente se una sottostringa della query è presente anche nella sequenza di riferimento, e quindi aggiornare il valore di $D(\cdot)$ di conseguenza.

L'algoritmo garantisce quindi di trovare tutti gli intervalli che consentono un massimo di z differenze, ma nella sua implementazione pratica in `BWA`, sono state introdotte diverse ottimizzazioni per migliorare ulteriormente le prestazioni e l'accuratezza.

Precalcolo

- Calcola la stringa BWT B per la stringa di riferimento X.
- Calcola gli array $C(\cdot)$ e $OCC(\cdot,\cdot)$ da B.
- Calcola la stringa BWT B' per il riferimento inverso.
- Calcola l'array $OCC'(\cdot,\cdot)$ da B'.

Procedure

```
InexactSearch(W, z)
  CalculateD(W)
  return InexRecur(W, |W|-1, z, 1, |X|-1)

CalculateD(W)
  k <- 1
  l <- |X|-1
  z <- 0
  for i = 0 to |W|-1 do
    k <- C(W[i]) + OCC'(W[i], k-1) + 1
    l <- C(W[i]) + OCC'(W[i], l)
    if k > l then
      k <- 1
      l <- |X|-1
      z <- z + 1
    D(i) <- z

InexRecur(W, i, z, k, l)
  if z < D(i) then
    return 0
  if i < 0 then
    return {[k, l]}
  I <- 0
  *I <- I ∪ InexRecur(W, i-1, z-1, k, l)
  for each b ∈ {A, C, G, T} do
    k <- C(b) + OCC(b, k-1) + 1
    l <- C(b) + OCC(b, l)
    if k ≤ l then
      **I <- I ∪ InexRecur(W, i, z-1, k, l)
      if b = W[i] then
        I <- I ∪ InexRecur(W, i-1, z, k, l)
      else
        I <- I ∪ InexRecur(W, i-1, z-1, k, l)
  return I
```

SPIEGAZIONE DEL CODICE

- **Precalcolo**: Prima di iniziare la ricerca, vengono precalcolate alcune strutture dati essenziali: la stringa BWT (B) e l'array di occorrenze (OCC) per la sequenza di riferimento X, la stringa BWT (B') per la sequenza di riferimento inverso e l'array di occorrenze (OCC') per quest'ultima.
- **InexactSearch(W, z)**: Questa procedura avvia la ricerca inesatta. Prende in input la sequenza di query W e il numero massimo di differenze consentite z. Inizia calcolando l'array D con la procedura `CalculateD`, e successivamente richiama la procedura ricorsiva `InexRecur` per trovare gli intervalli SA delle corrispondenze.
- **CalculateD(W)**: Questa procedura calcola l'array $D(\cdot)$ che limita la ricerca. L'array D memorizza per ogni posizione i nella sequenza di query il limite inferiore del numero di differenze presenti nel prefisso $W[0,i]$. La procedura utilizza la stringa BWT B' e l'array OCC' del riferimento inverso per verificare se una sottostringa di W è presente anche nel riferimento. Se una sottostringa non è presente,

il valore di z viene incrementato e il valore corrispondente in D viene aggiornato.

- **InexRecur(W, i, z, k, l)**: Questa procedura ricorsiva esplora lo spazio di ricerca. I parametri i, z, k e l rappresentano rispettivamente la posizione corrente nella sequenza di query, il numero di differenze accumulate, e l'intervallo SA corrente.
  - Se il numero di differenze accumulate z è minore del limite inferiore $D(i)$, la ricerca in quel ramo viene interrotta (non può portare a un allineamento valido).
  - Se $i < 0$, significa che si è arrivati alla fine della sequenza di query e l'intervallo SA corrente rappresenta una corrispondenza.
  - Le righe marcate con * e ** gestiscono rispettivamente le inserzioni e le delezioni nella sequenza di riferimento.
  - Per ogni possibile base b, la procedura calcola il nuovo intervallo SA e richiama ricorsivamente se stessa per esplorare il nuovo ramo.

OSSERVAZIONI

- L'algoritmo della sezione 4 rappresenta la base dell'allineamento inesatto in BWA, ma il programma implemento dai ricercatori nello studio [1] fornisce ulteriori ottimizzazioni, come l'uso di una struttura dati simile a un heap per gestire i risultati parziali e una strategia iterativa per accelerare la ricerca.
- La comprensione di questo algoritmo è fondamentale per analizzare le prestazioni di BWA e per comprendere come gestisce i mismatch e i gap durante l'allineamento delle *short reads*.

4.1 Ottimizzazioni pratiche dell'algoritmo

L'algoritmo analizzato nella precedente sezione 4, pur essendo completo in teoria, viene modificato nella pratica per migliorare le prestazioni di BWA. Le modifiche principali sono quattro:

(1) **Penalità differenziate:**
Invece di considerare tutte le differenze (mismatch, gap opening e gap extension) allo stesso modo, BWA assegna loro penalità diverse, rendendo l'allineamento più realistico dal punto di vista biologico.

(2) **Struttura dati a heap e ricerca breadth-first:**
BWA utilizza una struttura dati simile a un heap per memorizzare i risultati parziali (hit parziali), invece di usare la ricorsione. Questa struttura dà priorità ai risultati con punteggio di allineamento migliore, garantendo che BWA trovi sempre prima gli intervalli migliori. Inoltre, la sequenza letta complementata in modo inverso viene elaborata contemporaneamente. Questo approccio, basato su una ricerca breadth-first (BFS), si discosta dalla ricerca depth-first (DFS) simulata dalla ricorsione descritta nell'algoritmo analizzato nella sezione 4.

(3) **Strategia iterativa:**
BWA adotta una strategia iterativa per accelerare la ricerca. Se l'intervallo in cima all'heap (quello con il punteggio migliore) è ripetitivo, la ricerca di intervalli subottimali viene interrotta. Se invece l'intervallo è unico e ha una differenza z, la ricerca prosegue solo per hit con un massimo di $z + 1$ differenze. Questa strategia consente di accelerare BWA mantenendo la capacità di generare un punteggio di qualità dell'allineamento (mapping quality). Tuttavia, rende la velocità di BWA sensibile al tasso di mismatch tra le letture e il riferimento, poiché la ricerca di hit con più differenze è generalmente più lenta.

(4) **Seed sequence:**
BWA consente di impostare un limite al numero massimo di differenze consentite nelle prime decine di basi di una lettura, chiamate *seed sequence*. Ad esempio, è stato constatato dai ricercatori che con letture simulate di 70 bp, l'allineamento con un massimo di due differenze nella seed sequence di 32 bp è 2,5 volte più veloce rispetto a quello senza seeding. Il tasso di errore di allineamento, definito come la frazione di allineamenti errati tra gli allineamenti sicuri nella simulazione, aumenta solo dallo 0,08% allo 0,11%. Il seeding è meno efficace per le letture più brevi [1].

4.2 Gestione della memoria in BWA

L'algoritmo descritto nella sezione 4 richiede di caricare in memoria due strutture fondamentali: l'array delle occorrenze OCC e l'array dei suffissi SA. Tuttavia, mantenere interamente in memoria questi array può risultare molto dispendioso in termini di spazio.

Fortunatamente, è possibile ridurre il consumo di memoria conservando solo una parte degli array OCC e SA e calcolando il resto al momento del bisogno. Strategie simili vengono adottate anche da strumenti come *BWT-SW* e *Bowtie*.

Per un genoma di lunghezza n, l'array delle occorrenze $O(\cdot,\cdot)$ richiede $4n \log_2 n$ bit di memoria, poiché ogni valore occupa $\log_2 n$ bit e ci sono $4n$ valori nell'array. In BWA, invece di memorizzare l'intero array, vengono conservati in memoria solo i valori di $O(\cdot, k)$, dove k è un multiplo di 128. Gli altri valori vengono calcolati utilizzando la stringa BWT B. Inoltre, rappresentando ogni nucleotide con due bit, la stringa B occupa $2n$ bit. In totale, la memoria necessaria per la ricerca all'indietro è pari a $2n + n \log_2 n/32$ bit.

Poiché è necessario anche memorizzare il BWT del genoma inverso per calcolare i limiti, la memoria complessiva per il calcolo degli intervalli raddoppia, circa 2,3 GB per un genoma di 3 Gb.

Per identificare la posizione di ogni occorrenza di una stringa nel genoma, è necessario utilizzare l'array dei suffissi S. Memorizzare l'intero array S in memoria richiederebbe $n \log_2 n$ bit, dove n è la dimensione del genoma. Tuttavia, BWA utilizza una tecnica efficiente per ricostruire l'intero array S memorizzando solo una parte di esso. L'approccio di BWA si basa sull'array inverso dei suffissi compressi (*CSA inverso*) Ψ^{-1} e sull'array S. La relazione tra S e Ψ^{-1} è definita dalla seguente equazione:

$$S(k) = S((\Psi^{-1})^{(j)}(k)) + j \qquad (5)$$

dove $(\Psi^{-1})^{(j)}(k)$ indica l'applicazione ripetuta della trasformazione Ψ^{-1} per j volte. Il CSA inverso Ψ^{-1} può essere calcolato utilizzando l'array delle occorrenze OCC:

$$\Psi^{-1}(i) = C(B[i]) + O(B[i], i) \qquad (6)$$

dove:

(1) $B[i]$ è l'i-esimo carattere nella stringa BWT del genoma.

(2) $C(a)$ è il numero di caratteri nel genoma che sono lessicograficamente minori di a.

BWA memorizza in memoria solo gli elementi S(k) dell'array dei suffissi per i valori di k divisibili per 32. Per i valori di k che non sono divisibili per 32, BWA applica ripetutamente la trasformazione Ψ^{-1} fino a quando, per un certo valore j, $(\Psi^{-1})^{(j)}(k)$ risulta divisibile per 32. A questo punto, BWA può recuperare il valore di $S((\Psi^{-1})^{(j)}(k))$ dalla memoria e calcolare S(k) utilizzando l'equazione (5).

5 IMPLEMENTAZIONE E RISULTATI

Gli autori Li et al. [1] hanno sviluppato BWA per eseguire

l'allineamento di short read basandosi sulla Burrows-Wheeler Transform del genoma di riferimento. BWA gestisce l'allineamento con gap per le letture single-end, supporta la mappatura paired-end, genera punteggi di qualità di mappatura e può fornire più hit se necessario. Il formato di output predefinito è SAM (Sequence Alignment/Map), permettendo l'utilizzo di SAMtools per analisi downstream come l'estrazione di allineamenti in una regione specifica, il merge e l'ordinamento degli allineamenti, l'ottenimento di chiamate di polimorfismi a singolo nucleotide (SNP) e indel, e la visualizzazione degli allineamenti.

Gli autori valutano le prestazioni di BWA confrontandolo con altri tre programmi di allineamento: MAQ, SOAPv2 e Bowtie. Inizialmente, utilizzano dati simulati dal genoma umano per valutare l'accuratezza dell'allineamento. I risultati mostrano che BWA e MAQ ottengono un'accuratezza simile, con BWA leggermente più preciso di Bowtie e SOAPv2. In termini di velocità, SOAPv2 si dimostra il più veloce, seguito da BWA, che è significativamente più veloce di MAQ. Successivamente, gli autori utilizzano dati reali di sequenziamento Illumina per valutare le prestazioni dei programmi su dati reali. In questo caso, BWA e MAQ mostrano un'elevata percentuale di mappature corrette e coerenti con l'informazione di pairing. BWA si conferma più veloce di MAQ, mentre SOAPv2, pur essendo il più veloce, presenta una percentuale leggermente inferiore di mappature corrette. Infine, per valutare ulteriormente l'accuratezza, gli autori allineano le reads a un genoma ibrido uomo-pollo. BWA si dimostra accurato, mappando solo una piccola frazione di reads al genoma sbagliato. Questa analisi su dati simulati e reali evidenzia che BWA offre un buon compromesso tra velocità e accuratezza, risultando un valido strumento per l'allineamento di short reads.

Valutazione di BWA su Dati Reali

Gli autori si concentra sulla valutazione delle prestazioni di BWA su dati reali, utilizzando un approccio innovativo per valutare l'accuratezza dell'allineamento. Scaricano circa 12.2 milioni di coppie di reads da 51 bp da European Read Archive (sequenziate per l'individuo NA12750, incluso nel progetto 1000 Genomes). Le reads vengono mappate sul genoma umano (NCBI build 36). I risultati mostrano che quasi tutte le mappature considerati affidabili (*confident*) da MAQ e BWA sono presenti in coppie consistenti, anche se MAQ produce un numero inferiore di allineamenti confident rispetto a BWA. Una modalità di BWA più lenta (senza seeding e con ricerca di hit subottimali anche se il top hit è una ripetizione) ha ottenuto risultati ancora migliori. In questa modalità, BWA ha mappato in modo confident l'89.2% di tutte le reads in 6.3 ore, con il 99.2% delle mappature confident presenti in coppie consistenti.

Per valutare ulteriormente l'accuratezza degli allineatori, gli autori utilizzano un genoma ibrido uomo-pollo. Questo approccio permette di stimare la percentuale di reads mappate erroneamente al genoma sbagliato, fornendo un'indicazione dell'accuratezza

dell'allineamento. I risultati mostrano che BWA mappa solo una piccola percentuale di reads al genoma di pollo (2942 reads su 12.2 milioni), con un tasso di errore stimato dello 0.06%. Gli autori sottolineano che questa stima del tasso di errore potrebbe essere sottostimata o sovrastimata a causa di fattori come la presenza di sequenze altamente conservate tra uomo e pollo, l'assemblaggio incompleto del genoma umano o errori nell'assemblaggio del genoma di pollo.

È stata dimostrata l'efficacia di BWA nell'allineamento di dati reali, evidenziando la sua capacità di produrre mappature accurate e di gestire efficacemente reads paired-end. L'utilizzo di un genoma ibrido uomo-pollo rappresenta un metodo interessante per valutare l'accuratezza degli allineatori, offrendo una prospettiva aggiuntiva rispetto alle simulazioni e alle metriche basate sulla consistenza del pairing.

6 CONCLUSIONI

BWA rappresenta un significativo passo avanti nel campo

dell'allineamento di short reads, dimostrandosi un ordine di grandezza più veloce rispetto a MAQ, pur mantenendo un buon equilibrio tra velocità, consumo di memoria e accuratezza

dell'allineamento. Supportando l'allineamento con gap per letture single-end, BWA si adatta bene all'evoluzione delle tecnologie di sequenziamento, dove le letture più lunghe tendono a includere indel. L'emissione degli allineamenti nel formato SAM consente di sfruttare appieno gli strumenti downstream disponibili in SAMtools per analisi avanzate.

Tuttavia, la precisione dell'allineamento su dati reali resta difficile da valutare in modo oggettivo, specialmente in scenari complessi con letture lunghe o elevate percentuali di variabilità genomica. Le prestazioni di BWA degradano quando la lunghezza delle letture aumenta, principalmente a causa della necessità di allineare interamente ogni lettura, dalla prima base all'ultima. Per affrontare questa limitazione, si potrebbe adottare una strategia basata sulla frammentazione delle letture: dividendo le letture lunghe in più frammenti corti, è possibile allinearli separatamente e combinare gli allineamenti parziali per ricostruire l'allineamento completo. Questa soluzione, unita alle caratteristiche esistenti di BWA, potrebbe ulteriormente migliorare la sua efficienza e applicabilità in analisi genomiche avanzate.

REFERENCES

[1] H. Li, "Fast and accurate short read alignment with burrows-wheeler transform," *Bioinformatics (Oxford, England)*, vol. 25, pp. 1754–60, 06 2009.

[2] A. D. Smith, Z. Xuan, and M. Q. Zhang, "Using quality scores and longer reads improves accuracy of solexa read mapping," *BMC Bioinformatics*, vol. 9, no. 1, p. 128, 2008. [Online]. Available: https://doi.org/10.1186/1471-2105-9-128

[3] H. Li, J. Ruan, and R. Durbin, "Mapping short dna sequencing reads and calling variants using mapping quality scores," *Genome Research*, vol. 18, no. 11, pp. 1851–1858, Nov 2008, epub 2008 Aug 19.

[4] H. Lin, Z. Zhang, M. Q. Zhang, B. Ma, and M. Li, "ZOOM! Zillions of oligos mapped," *Bioinformatics*, vol. 24, no. 21, pp. 2431–2437, Nov 1 2008, epub 2008 Aug 6.

[5] H. Jiang and W. H. Wong, "SeqMap: mapping massive amount of oligonucleotides to the genome," *Bioinformatics*, vol. 24, no. 20, pp. 2395–2396, Oct 15 2008, epub 2008 Aug 12.

[6] M. C. Schatz, "CloudBurst: highly sensitive read mapping with MapReduce," *Bioinformatics*, vol. 25, no. 11, pp. 1363–1369, 04 2009. [Online]. Available: https://doi.org/10.1093/bioinformatics/btp236

[7] R. Li, Y. Li, K. Kristiansen, and J. Wang, "Soap: Short oligonucleotide alignment program," *Bioinformatics (Oxford, England)*, vol. 24, pp. 713–4, 04 2008.

[8] D. J. Campagna, A. Albiero, A. Bilardi, E. Caniato, C. Forcato, S. Manavski, N. Vitulo, and G. Valle, "Pass: a program to align short sequences," *Bioinformatics*, vol. 25 7, pp. 967–8, 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:13568115

[9] H. L. Eaves and Y. Gao, "MOM: maximum oligonucleotide mapping," *Bioinformatics*, vol. 25, no. 7, pp. 969–970, Apr 1 2009, epub 2009 Feb 19.

[10] N. Malhis, Y. Butterfield, M. Ester, and S. Jones, "Slider–maximum use of probability information for alignment of short sequence reads and snp detection," *Bioinformatics*, vol. 25, pp. 6–13, 01 2009.

[11] S. Kumar, S. Agarwal, and R. Prasad, "Efficient read alignment using burrows wheeler transform and wavelet tree," in *2015 Second International Conference on Advances in Computing and Communication Engineering*, 2015, pp. 133–138.

[12] K. M. Robinson, A. S. Hawkins, I. Santana-Cruz, R. S. Adkins, A. C. Shetty, S. Nagaraj, L. Sadzewicz, L. J. Tallon, D. A. Rasko, C. M. Fraser, A. Mahurkar, J. C. Silva, and J. C. D. Hotopp, "Aligner optimization increases accuracy and decreases compute times in multi-species sequence data," *Microbial Genomics*, vol. 3, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:7257799

[13] Y. Jung and D. Han, "Bwa-meme: Bwa-mem emulated with a machine learning approach," 09 2021.

[14] C. Kim, K. Koh, T. Kim, D. Han, and J. Seo, "Bwa-mem-scale: Accelerating genome sequence mapping on commodity servers," in *Proceedings of the 51st International Conference on Parallel Processing*, ser. ICPP '22. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3545008.3545033

[15] H. Li, "Aligning sequence reads, clone sequences and assembly contigs with bwa-mem," 2013. [Online]. Available: https://arxiv.org/abs/1303.3997

[16] D. E. Knuth, J. H. Morris, and V. R. Pratt, "Fast pattern matching in strings," *SIAM J. Comput.*, vol. 6, pp. 323–350, 1977. [Online]. Available: https://api.semanticscholar.org/CorpusID:11697579

[17] D. Gusfield, "Algorithms on strings, trees, and sequences - computer science and computational biology," 1997. [Online]. Available: https://api.semanticscholar.org/CorpusID:61800864

[18] M. Farach, "Optimal suffix tree construction with large alphabets," in *Proceedings 38th Annual Symposium on Foundations of Computer Science*, 1997, pp. 137–143.

[19] Anonymous, "Burrows-wheeler transform," accessed: 2024-11-02. [Online]. Available: https://www.cs.cmu.edu/~15451-f18/lectures/lec25-bwt.pdf

[20] P. Ferragina and G. Manzini, "Indexing compressed text," *J. ACM*, vol. 52, no. 4, p. 552–581, Jul. 2005. [Online]. Available: https://doi.org/10.1145/1082036.1082039

[21] G. Urgese, G. Paciello, A. Acquaviva, E. Ficarra, M. Graziano, and M. Zamboni, "Dynamic gap selector: A smith waterman sequence alignment algorithm with affine gap model optimisation." 04 2014.