Allineamento di sequenze biologiche: BLAST e le sue varianti

Ciro e Vincenzo Malafronte

January-February 2024

Sommario

1 Introduzione

Il seguente lavoro è stato svolto come progetto per un esame universitario relativo gli Strumenti formali della Bioinformatica. La bioinformatica è una disciplina nuova, inizialmente si occupava principalmente dello studio del DNA e RNA, poi con lo sviluppo delle tecnologie ha portato ad un vasto uso dell'informatica in molti settori della biologia, il che ha portato poi a coniare il nuovo termine, ormai universalmente accettato, di Biologia Computazionale che esplicita con maggior chiarezza e precisione i reali e più vasti contenuti scientifici e disciplinari del connubio tra informatica e biologia. Quando si sequenzia un nuovo gene o una nuova proteina, non è sempre immediato capire quale sia la sua funzione biologica. Tuttavia, se durante l'analisi si rileva un'omologia con una proteina già conosciuta, è possibile formulare alcune ipotesi sulla funzione del nuovo gene. Nel corso del tempo sono stati sviluppati numerosi tools per la ricerca delle sequenze simili tra loro all'interno di un database, la maggior parte dei quali utilizzano delle misure di similarità tra le sequenze per poter definire delle relazioni biologiche significativamente simili tra campioni diversi. I metodi utilizzati in arte per queste ricerche utilizzano un connubio tra le misure di similarità e gli algoritmi di programmazione dinamica. Questi metodi richiedono una grossa potenza computazionale il che li rende impraticabili per le ricerche su grandi database, soprattutto se non si è in possesso di super computer. Con il proseguire degli studi sono stati sviluppati nuovi metodi e nuovi software che cercano di utilizzare i metodi sopra descritti per effettuare ricerche su database di grandi dimensioni utilizzando hardware meno potenti. Tra i software più popolari troviamo FASTP, un software che cerca prima delle regioni simili tra due sequenze senza valutare le identità, quindi riassegna i punteggi di queste sequenze utilizzando delle misure di similarità sulle regioni analizzate come matrice PAM, la quale permette di conservare la propria identità e, allo stesso tempo, di incrementare

le misure di similarità. Durante il corso dello studio è stato analizzato un altro metodo, più "moderno" e anch'esso molto utilizzato, chiamato BLAST(Basic Alignment Search Tool) che sviluppa una misura basata sui "punteggi di mutazione ben definiti". Questo metodo è in grado di rilevare una sequenza debole ma con somiglianze biologicamente significative ed è di un ordine di grandezza più veloce rispetto agli altri algoritmi euristici esistenti.

2 Basic Logical Alignment Search Tool (BLAST)

BLAST è uno dei software più potenti in bioinformatica, ampiamente apprezzato tra i ricercatori per confrontare sequenze biologiche. Questo strumento consente ai ricercatori di confrontare una sequenza di loro interesse con un vasto archivio di sequenze già conosciute. Attraverso questa analisi, è possibile individuare le sequenze nel database che presentano somiglianze con quella di interesse. BLAST trova applicazione in molteplici contesti bioinformatici, inclusi l'identificazione di geni e proteine, l'analisi del genoma e lo studio dell'evoluzione molecolare. Grazie alla sua efficacia e flessibilità, BLAST è diventato uno strumento indispensabile per i ricercatori nell'analisi delle sequenze biologiche.

2.1 Scoring-matrix: PAM vs BLOSUM

Prima di introdurre l'algoritmo di BLAST è doveroso fare un passo indietro ed illustrare lo strumento fondamentale sul quale si basano le sue funzioni: **Le matrici di similarità**. Le matrici di similarità sono uno strumento fondamentale per valutare quello che è l'*indice di similarità* tra due o più sequenze biologiche, ovvero la probabilità che due elementi di due sequenze distinte siano state generate da uno stesso antenato. Il calcolo di questi indici è fondamentale per programmi come BLAST e, più in generale, per i programmi che lavorano sulla base di un allineamento delle sequenze in analisi, per poter valutare la qualità dell'allineamento tra due sequenze genetiche prima di passare all'analisi vera e propria. Le matrici di similarità sono costruite sulla base di modelli statistici che riflettono la probabilità di osservare specifiche sostituzioni tra gli elementi delle sequenze durante il processo evolutivo. Il funzionamento delle matrici di similarità è molto sem-

plice: ricevono due sequenze biologiche (nel nostro caso) e le allineano, dopodiché applicano un punteggio ad ogni coppia x_i, y_i composta dall' i-esimo elemento della prima sequenza e dall' i-esimo elemento della seconda sequenza. Il punteggio, chiamato anche punteggio di sostituzione, viene applicato in base alla probabilità che i due elementi, solitamente nucleotidi o amminoacidi, abbiano un genitore in comune. In questo modo, la matrice fornisce un indice di similarità che tiene conto della natura biologica delle sequenze, attribuendo punteggi più alti a coppie di elementi simili e punteggi più bassi a coppie di elementi diversi. Le matrici di similarità più note e utilizzate sono :

- **PAM (Point Accepted Mutation)**, sviluppate da Margaret Dayhoff negli anni '70 e '80, si basano su modelli evolutivi di mutazione degli amminoacidi. Esse sono espresse in unità PAM e classificate in base al numero di PAM a cui corrispondono le mutazioni rappresentate nella matrice. Le matrici PAM sono particolarmente adatte per l'allineamento di sequenze molto simili, poiché sono costruite utilizzando dati di allineamento di sequenze con un'omologia significativa. Tra le matrici PAM più utilizzate ci sono:
  - **PAM250**: rappresenta le sostituzioni che si verificano ogni 250 PAM e viene utilizzata per l'allineamento di sequenze particolarmente divergenti;
  - **PAM120**: rappresenta le sostituzioni che si verificano in media ogni 120 PAM. È utilizzata per l'allineamento di sequenze con un grado di similarità moderato;
  - **PAM70**: rappresenta le sostituzioni che si verificano in media ogni 70 PAM. È adatta per l'allineamento di sequenze molto simili tra loro.
- **BLOSUM (BLOcks SUbstitution Matrix)**, sviluppate da Steven Henikoff e Jorja Henikoff negli anni '90, si basano sull'analisi

di blocchi conservati di sequenze amminoacidiche. Queste matrici sono denominate con numeri che indicano la percentuale di sostituzioni osservate nei blocchi di allineamento utilizzati per costruire la matrice (ad esempio, BLOSUM62). Queste sono preferite quando si allineano sequenze più divergenti, poiché riflettono meglio le sostituzioni amminoacidiche conservate nei blocchi di sequenze omologhe. Tra le matrici BLOSUM più conosciute abbiamo:

- **BLOSUM45**: Questa matrice è più conservativa rispetto a BLOSUM62 e viene spesso utilizzata per allineare sequenze molto simili.
- **BLOSUM50**: Anche questa è una matrice relativamente conservativa, ma meno di BLOSUM45. È adatta per l'allineamento di sequenze con un grado moderato di divergenza.
- **BLOSUM62**: Questa è la matrice di default utilizzata da BLAST. È una delle varianti più comuni e viene utilizzata per l'allineamento di sequenze con un grado moderato di divergenza.
- **BLOSUM80**: Questa matrice è più aggressiva rispetto a BLOSUM62 e viene utilizzata per allineare sequenze più divergenti.
- **BLOSUM90**: Questa è una delle matrici più aggressive e viene utilizzata per allineare sequenze molto divergenti.

La scelta della matrice da utilizzare dipende molto dal grado di divergenza tra le sequenze da confrontare, ovvero il numero di mutazioni e cambiamenti che determinati nucleotidi hanno subito nel corso del tempo. Inoltre tale scelta è cruciale per ottenere risultati accurati e significativi durante l'analisi delle sequenze proteiche. Ad esempio se sappiamo che le sequenze da confrontare appartengono ad orizzonti

evolutivi differenti delle matrici BLOSUM62 o BLOSUM50 sono considerate più adatte poiché sono più "profonde" rispetto ad una matrice PAM30 che, seppur considerata profonda tra le matrici PAM, è più superficiale rispetto alle BLOSUM. Il motivo di questa differenza ed anche predilezione, se così si può considerare, per le matrici BLOSUM sta nella loro costruzione. Esse sono infatti costruite analizzando blocchi di sequenze aminoacidiche che sono rimaste invariate o quasi nel corso del tempo. Questa particolarità costruttiva permette alle matrici BLOSUM di offrire una maggiore flessibilità nel corso delle analisi che per sequenze particolarmente divergenti si trasforma in maggiore accuratezza. Ciò nonostante per confronti tra sequenze poco divergenti le matrici PAM sono ancora preferite nel panorama biologico poiché sono più ottimizzati per quel tipo di analisi. Il software BLAST permette di scegliere quale tipo di matrice utilizzare attraverso la specifica `-m` oppure `-matrix` ma di default utilizza le matrici BLOSUM62.

2.2 L'algoritmo BLAST

L'algoritmo parte da un input che equivale alla query che si vuole fare al database, ovvero la sequenza biologica che vogliamo analizzare. Partendo dall'input abbiamo una fase preliminare che consiste nel dividere la query in varie sotto stringhe di lunghezza W, definite *w-mers* o anche *seed*. Dopodiché si procede nell'ordine seguente:

- **Scanning del DB:** le w-mers vengono confrontate con le sequenze presenti nel DB; per ogni sequenza viene calcolato un indice di similarità attraverso un'apposita matrice di similarità, dopodiché viene costruita una lista con tutte le sequenze del database che hanno un indice di similarità che sia almeno uguale ad una soglia minima T

- **Estensione degli hit:** Una volta costruita la lista di sequenze candidate basate sull'indice di similarità iniziale, si procede es-

tendendo gli "hit" o le corrispondenze iniziali per includere regioni adiacenti con punteggi di allineamento significativi. Questo processo si svolge tipicamente in due fasi:

1. **Prolungamento locale degli allineamenti:** Le corrispondenze iniziali vengono estese localmente lungo le sequenze del database e la query. L'obiettivo è allungare l'allineamento in modo da includere le regioni adiacenti che presentano somiglianze significative, mantenendo al contempo un punteggio di allineamento elevato. Questo processo si basa sull'utilizzo di algoritmi euristiche, come ad esempio l'algoritmo di Smith-Waterman per allineamenti locali, che consentono di identificare le estensioni ottimali dell'allineamento.
2. **Unione degli allineamenti locali:** Dopo aver prolungato localmente gli allineamenti lungo le sequenze, si procede unendo gli allineamenti locali sovrapposti o vicini. Questo processo aiuta a combinare diverse regioni di somiglianza che potrebbero essere state identificate separatamente, portando a un allineamento più completo e significativo tra la query e le sequenze del database.

- **Valutazione dei punteggi di allineamento:** Durante e dopo l'estensione degli hit, i punteggi di allineamento vengono valutati utilizzando una scoring matrix. Questi punteggi tengono conto delle corrispondenze, dei mismatch e degli spazi vuoti (gap) tra le sequenze allineate. Questa valutazione permette di determinare la significatività degli allineamenti identificati e di selezionare quelli più rilevanti e significativi per l'analisi successiva.
- **Filtraggio dei risultati:** Infine, si effettua una selezione accurata dei risultati degli allineamenti basandosi su diversi criteri, come il punteggio di allineamento ottenuto, la lunghezza

dell'allineamento e altri fattori rilevanti. Questo processo permette di scartare gli allineamenti casuali o poco significativi e di concentrarsi esclusivamente su quelli che hanno maggiori probabilità di essere biologicamente rilevanti per l'analisi in corso.

2.3 I metodi di ottimizzazione

Fasi come l'espansione e la scansione sono fasi delicate per un algoritmo perchè rischiano di fungere da collo di bottiglia e di appesantire particolarmente l'esecuzione dell'algoritmo; per questo motivo BLAST implementa una serie di ottimizzazioni volte proprio ad evitare il protrarsi indefinito dell'esecuzione:

- **Scansione ridotta del database (database indexing):** BLAST utilizza una struttura di dati indicizzata, come ad esempio un indice di hash, per velocizzare la scansione del database di sequenze. Questo consente di identificare rapidamente le regioni del database che potrebbero contenere corrispondenze con la query.

- **Allineamenti locali anziché globali:** BLAST effettua principalmente allineamenti locali invece di allineamenti globali, concentrandosi solo sulle regioni di somiglianza significative tra la query e le sequenze del database. Quest'operazione aiuta a ridurre il tempo computazionale confrontando solo regioni rilevanti delle sequenze anziché l'intera lunghezza delle sequenze.

- **Estensione degli allineamenti locali:** BLAST estende gli allineamenti locali solo nelle regioni più promettenti, utilizzando tecniche euristiche per determinare l'estensione ottimale dell'allineamento senza esaminare l'intera lunghezza delle sequenze coinvolte. Inoltre, viene calcolata una soglia S e quando la differenza tra il punteggio della coppia precedente e quello della coppia appena generata è maggiore di S, l'espansione viene bloccata.

Con il rilascio della versione `2.15.0` di BLAST sono stati aggiunti due metodi basati sulla parallelizzazione per ottimizzare le ricerche.

- **Thread by query:** Questo approccio si adatta bene ai casi in cui ci sono molte query da processare e il database è di dimensioni relativamente contenute. Con questa strategia, ogni thread riceve un gruppo di query, esegue la ricerca nell'intero database e formatta i risultati. Successivamente, il thread controlla se ci sono ulteriori query da elaborare. L'efficacia di questo metodo in termini di velocità dipende dal tipo di programma utilizzato (es. BLASTP, BLASTN) e dalla natura del task da svolgere (es. blastp-fast).

- **Thread by database:** Questo approccio è efficace per database di dimensioni ampie e gestisce facilmente qualsiasi quantità di query. In pratica, tutti i thread si occupano di un gruppo di query ciascuno, suddividendo il compito di ricerca nel database.

A parità di input l'output non cambia in base al metodo che viene utilizzato, ciò nonostante in base alla situazioine un metodo può risultare più performante dell'altro. BLAST sceglie automaticamente il metodo migliore in base alla grandezza del DB e delle query da analizzare. Questa scelta può anche essere sovrascritta andando ad imporre un determinato modello ma gli stessi sviluppatore del software scoraggiano tale scelta.

3 MegaBLAST vs HS-BLASTn

3.1 MegaBLAST

MegaBLAST è attualmente il modulo predefinito chiamato dal programma NCBI-BLASTN che ha come scopo quello di sfruttare la parallelizzazione come metodo principale. Col tempo sono nate ulteriori tecniche volte a velocizzare MegaBLAST, puntando ad avere un incremento di velocità producendo però gli stessi risultati del genitore. Le implementazioni più note e performanti sono:

- **Indexed MegaBLAST:** accelera MegaBLAST costruendo un indice per il database di destinazione. Memorizza le posizioni di ogni k-mer (dove $w \geq k$) che termina in ogni s-esima posizione (s = w - k + 1) nel database. L'algoritmo di ricerca dei seed identifica prima i k-seed utilizzando l'indice, e quindi controlla se ogni k-seed è contenuto in un w-seed. Gli esperimenti condotti hanno mostrato che questa tecnica è dalle 2 alle 4 volte più veloce di MegaBLAST. Tuttavia, la procedura di controllo può richiedere tempo quando le query sono lunghe e il database è grande.

- **G-BLASTN:** L'altro strumento di ricerca, G-BLASTN, è un'alternativa open-source basata su GPU. Utilizza le GPU per parallelizzare la fase di scansione di NCBI-BLASTN. Rispetto a MegaBLAST sequenziale, G-BLASTN è 14,8 volte più veloce.

In MegaBLAST, il processo di ricerca avviene in tre fasi:

- **preparazione:** vengono configurate opzioni, query e database, e viene creata una lookup table per le query;

- **ricerca preliminare:** si esaminano i soggetti alla ricerca di "seed" utilizzando la lookup table e si eseguono allineamenti senza gap. Se superano una soglia, si attiva una ricerca con

gap. Gli allineamenti con gap, superata un'altra soglia, vengono salvati come corrispondenze preliminari;

- **traceback:** vengono considerati i nucleotidi ambigui dalle corrispondenze preliminari e restituiti con le informazioni di traceback, che includono il numero e le posizioni delle inserzioni, delle eliminazioni e delle corrispondenze.

La lookup table costruita dall'algoritmo MegaBLAST è una tabella hash che memorizza gli offset dei k-mer trovati nelle query. Durante la fase di ricerca preliminare, si esaminano i soggetti per trovare i seed. Ogni k-mer del soggetto viene confrontato con la tabella di lookup per individuare corrispondenze. Se necessario, si scansiona il database in passi definiti per trovare tutti i seed.

3.2 HS-BLASTN

HS-BLASTN è un algoritmo progettato per accelerare l'allineamento di sequenze biologiche utilizzando la Trasformata di Burrow-Wheeler (BWT), una tecnica di compressione dati molto usata nell'ambito delle *next-generation sequencing analysis*. Questo algoritmo crea una struttura di indicizzazione basata sulla BWT per mappare le sequenze del database, ottimizzando così la ricerca. Utilizzando questa struttura, HS-BLASTN accelera l'esecuzione di MegaBLAST senza compromettere i risultati dell'allineamento. Inoltre, HS-BLASTN sfrutta una nuova lookup table, diversa da quella costruita da MegaBLASt, derivata dal database e basata sull'FMD-index, che consente una ricerca accurata ed efficiente dei seed identificati da MegaBLAST. Questo rende HS-BLASTN particolarmente efficace quando si vogliono allineare un gran numero di query.

3.2.1 La lookup table

Come quella di MegaBLAST e di Indexed-MegaBLAST anche la lookup table di HS-BLASTN è hash table, con la differenza che ogni cella conserva le informazioni riguardanti il bi-interval di un k-mer. Dato un testo S di lunghezza n ed una stringa P di lunghezza m, il bi-interval di P in S consiste in una coppia di intervalli che indica la posizione di P e del suo complementare $\overline{P}$ all'interno del FMD-index di S. Attraverso il bi-interval si possono trovare facilmente i k-mer che iniziano o finiscono con P nel database. Il numero di entries nella lookup table è 4^k e presenta tre vantaggi:

1. Viene costruita una sola volta per poi essere utilizzata per ogni ricerca all'interno del database;

2. Occupa molto meno spazio rispetto alla lookup table di Indexed-MegaBLAST;

3. É più efficace delle lookup table usate da MegaBLAST ed Indexed-MegaBLAST.

3.2.2 Il nuovo algoritmo di seeding

L'algoritmo di seed search, o seeding, è il codice che si occupa di cercare all'interno del db tutti i w-seed delle query. Quello utilizzato da HS-BLASTN si svolge in due fasi:

1. **Bi-interval identification:** Come mostrato in figura 1 HS-BLASTN scandisce la query in passi (strides) di lunghezza w - k + 1, dove w è un altro parametro scelto dall'utente. Per ogni k-mer incontrato nella query, HS-BLASTN calcola il suo valore hash, che è un numero che identifica il k-mer, e lo usa per cercare il suo bi-interval nella tabella di ricerca, che è una

```
Input: query Q
Output: All the bi-intervals of w-seeds
1   scan_stride = w − k + 1;
2   initialize Match as an empty array;
3   i = 0;
4   while i <= |Q| - k do
5       initialize Prev as an empty array
6       index = HASH(Q[i, i + k − 1]);
7       [l, u, s] = LookupTable[index];
8       for j = i − 1 to 0 do
9           [l0, u0, s0] = BackwardExt([l, u, s], Q[j]);
10          if s0 < s then
11              Append [l, u, s, j − i + k] to Prev;
12          end if
13          if s0 == 0 or i − j + k == w then
14              break;
15          end if
16          [l, u, s] = [l0, u0, s0];
17      end for
18      for r = |Prev| - 1 to 0 do
19          [l, u, s, e] = Prev[r];
20          query_start = i − e + 1;
21          for j = i + k to |Q| - 1 do
22              if s == 0 or e == w then
23                  break;
24              end if
25              [l0, u0, s0] = ForwardExt([l, u, s]);
26              e = e + 1;
27              [l, u, s] = [l0, u0, s0];
28          end for
29          if e == w and s != 0 then
30              Append (query_start, [l, u, s]) to Match;
31          end if
32      end for
33      i = i + scan_stride;
34  end while
35  return Match;
```

Figure 1: **Identificazione dei bi-intervalli: il primo passo dell'algoritmo di ricerca dei seed in HS-BLASTN**

struttura dati che memorizza i bi-interval di tutti i possibili k-mer. Se il bi-interval esiste, significa che il k-mer è presente nel database. A questo punto, HS-BLASTN usa l'algoritmo di estensione all'indietro per cercare di allungare il k-mer a sinistra, aggiungendo una base alla volta. Se l'estensione all'indietro fallisce, significa che il k-mer non può essere parte di un seme più lungo in quella direzione. Quindi, HS-BLASTN usa l'algoritmo di estensione in avanti per cercare di allungare il k-mer a destra, fino a raggiungere la lunghezza minima di w. Se l'estensione in avanti riesce, significa che il k-mer è parte di un seme valido e il suo bi-interval viene salvato in una lista. Questo processo viene ripetuto per ogni k-mer nella query, fino a trovare tutti i

semi possibili. Questi semi saranno poi usati nel secondo passo dell'algoritmo, che consiste nell'allineare la query con il database usando una tecnica chiamata *dynamic programming*.

2. **Occurrence position detextion:** Questa fase è illustrata dall'immagine 2 e consiste nel trovare le posizioni esatte di tutti i semi trovati nel primo passo nel database. Per fare ciò utilizza il bi-interval di ogni seed per accedere al *suffix-array* (SA), che è una struttura dati che ordina tutti i suffissi del database. Tuttavia, la SA può essere troppo grande per stare nella memoria RAM, quindi HS-BLASTN usa una versione ridotta, che memorizza solo alcune posizioni a intervalli regolari. Questo intervallo(r) è un parametro che bilancia l'efficienza e l'uso della memoria. Se la posizione k è un multiplo di r, allora SA[k] è disponibile e può essere usato subito. Altrimenti viene usata una funzione chiamata LF-mapping per trovare SA[k] a partire da una posizione x che è un multiplo di r, aggiungendo il numero di iterazioni necessarie.

Input: Array of bi-intervals Match, suffix array interval r

Output: All the w-seeds

```
1   initialize Seeds as an empty array;
2   for each (query_start, [L, U, S]) in Match do
3       for k = L to L + S − 1 do
4           iter = 0;
5           x = k;
6           while x % r != 0 do
7               c = B[x];
8               x = C[c] + O[c][x];
9               iter = iter + 1;
10          end while
11          subject_start = SA[x] + iter;
12          Append (query_start, subject_start, w) to Seeds;
13      end for
14  end for
15  return Seeds;
```

Figure 2: Rilevamento della posizione di occorrenza: il secondo passo dell'algoritmo di ricerca dei seed in HS-BLASTN.

Analizzando la complessità di questo algoritmo possiamo vedere che al passo uno proviamo ad estendere in avanti e/o indietro la sequenza $\|Q\|$ volte, dove $\|Q\|$ è la lunghezza della query Q; le chiamate a BackwardExt e ForwardExt consistono nel chiamare LF-mapping 4 volte. Poichè LF-mapping viene risolto in tempo costante il primo step si risolve in tempo $\boldsymbol{O(\|Q\|)}$. Al secondo step, supponendo di computare s w-seeds, dobbiamo calcolare il tempo di prelevare s volte SA[k]. Poichè SA ha tempo di lettura costante, la complessità del secondo step è $\boldsymbol{O(s)}$.

3.2.3 L'implementazione del FMD-index

Per HS-BLASTN sono state effettuate delle modifiche al FMD-index rispetto alla sua versione originale. La prima modifica è che la lookup table è costruita per il database target ed è integrata nell'FMD-index. La seconda modifica consiste nel valore dell'intervallo della suffix array che è cambiato da 32 a 8. Quando una query è lunga e il database è grande, si otterranno molti bi-interval di w-seeds, e di conseguenza verranno eseguite più chiamate ad LF-mapping. Di conseguenza, le prestazioni di HS-BLASTN peggiorano. Per risolvere questo problema, si dovrebbe usare un valore più piccolo di r.

3.2.4 Parallelizzazione

Come si può vedere nella figura 3 per ottimizzare al meglio l'algoritmo e per sfruttare al pieno la potenza computazionale dell'hardware, HS-BLASTN può essere eseguito usando più thread. Gli utenti possono specificare un numero desiderato di thread di ricerca attraverso l'opzione *-num_threads*, come in BLASTN. HS-BLASTN prepara prima il database e le query, dopodichè distribuisce le query tra i thread di ricerca. Poichè le query sono indipendenti tra loro, questa divisione non impatta minimamente sul risultato. Ogni thread avvia il motore di ricerca in modo

concorrente. Infine, quando terminano i loro compiti, i vari thread si sincronizzano tra loro e il thread principale unisce tutti i risultati in un unico output.

```
Input: FMD-index of the database, queries
Output: Local alignments between the database and the queries
1   prepare options;
2   prepare FMD-index;
3   prepare queries;
4   distribute queries across N threads;
5   parallel do for all threads
6       find seeds using the FMD-index;
7       perform ungapped extension;
8       perform gapped extension;
9       perform traceback;
10  end for
11  synchronize;
12  merge alignments and output them;
```

Figure 3: Pseudocodice del funzionamento di HS-BLASTN con N threads

3.3 Esperimenti e risultati

Tutti gli esperimenti presentati di seguito sono stati condotti in ambiente WSL, acronimo di "Windows Subsystem for Linux"; una funzionalità di Windows che consente agli utenti di eseguire un ambiente Linux direttamente su un sistema Windows. L'intero sistema gira su un processore intel core i9-12900H con 20 cores e 16GB di RAM. Il confronto tra MegaBLASt ed HS-BLASTN è stato eseguito con le versioni 2.2.30+, per MegaBLAST, e 0.0.5 per HS-BLASTN. Abbiamo utilizzato una versione più leggera del database *human build 38*, db contenente sequenze di genomi umani, modificata per essere grande poco più di 1GB in modo da farlo girare sulla nostra macchina. L'FMD-index costruito è grande circa 8 GB ed HS-BLASTN impiega circa 42 secondi per caricarlo. Per quanti riguarda le query, invece, abbiamo usato due set di query:

- **HSQueriesSmall:** contenente 2 milioni di query con lunghezze dai 100 ai 500 caratteri;
- **HSQueriesLarge:** contenente circa 870000 query di lunghezza tra 800 e 4000 caratteri.

Abbiamo eseguito sia MegaBLAST che HS-BLASTN utilizzando diversi set di query e variando il numero di thread per ognuno di essi. I tempi sono stati calcolati attraverso il comando `time` di Linux. La figura 3.3 mostra i tempi di HS-BLASTN e MegaBLAST e lo speedup calcolato come TH/TM. Come si può notare le prestazioni di HS-BLASTN sono fino a 16 e 17 volte superiori a quelle di MegaBLAST nonostante l'esecuzione sia eseguita su una WSL. Nonostante entrambi gli algoritmi migliorino con l'aumentare del numero di thread utilizzati si può vedere come HS-BLASTN scali meglio del suo concorrente ed abbia tempi nettamente migliori.

HSQueriesSmall			
THREADS	**MegaBLAST**	**HS-BLASTN**	**SPEEDUP**
1	3099,50	373,59	8,00
4	1854,05	134,78	13,00
8	1256,96	77,75	16,00
HSQueriesLarge			
THREADS	**MegaBLAST**	**HS-BLASTN**	**SPEEDUP**
1	5864,44	562,31	10,00
4	3022,23	184,50	16,00
8	1745,74	100,82	17,00

4 Conclusioni

Abbiamo introdotto HS-BLASTN, uno strumento per la ricerca di sequenze di DNA, che offre un notevole miglioramento delle prestazioni rispetto a MegaBLAST. HS-BLASTN accelera MegaBLAST creando una tabella di ricerca che memorizza informazioni su sequenze di lunghezza specifica e utilizzando un metodo di seeding che esamina le query a intervalli regolari. Attraverso prove sperimentali, abbiamo dimostrato che HS-BLASTN supera MegaBLAST in termini di velocità e offre una migliore efficienza in esecuzione parallela. Considerando l'espansione delle sequenze biologiche, HS-BLASTN rappresenta un significativo progresso nella ricerca genomica grazie alla sua notevole velocità migliorata rispetto a MegaBLAST.