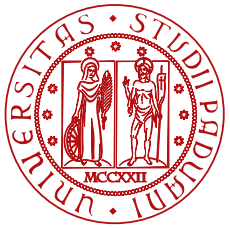

Università degli Studi di Padova

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELLINFORMAZIONE
Corso di Laurea in Ingegneria Informatica

Presentazione e confronto di metodi per l'archiviazione efficiente di dati per la bioinformatica

Relatore

Prof. Comin Matteo

Università degli studi di Padova

Laureando

Umberto Salviati

Mat. 1220994

Anno Accademico 2021-2022
Data di laurea 23 settembre

Alla mia famiglia...

Abstract

Negli ultimi decenni sono state sviluppate nuove tecnologie che permettono di informatizzare i genomi studiati in laboratorio. I dati sequenziati, però, stanno rapidamente riempiendo i database pubblici. È quindi scopo di molti tool moderni quello di salvare in modo efficiente questi dati. In queste analisi andremo a confrontare due tool promettenti per la compressione di questi dati. Questi tool operano sui k-mer, strumento pensato per l'analisi dei file fasta, output delle nuove tecniche di sequenziamento. Come vedremo, tra il metodo Counting de Bruijn Graph e il tool UST, sarà quest'ultimo a rivelarsi il più efficiente a salvare i dati genetici. Queste analisi, però, hanno il solo scopo di confrontare la dimensione dei file in output e non considerano altri aspetti di questi tool. È quindi corretto affermare che questa analisi sia parziale.

Indice

Capitolo 1

Introduzione

1.1 Genomica

1.1.1 DNA

L'acido desossiribonucleico (in sigla DNA) è un acido nucleico che contiene le informazioni genetiche necessarie alla biosintesi di RNA e proteine, molecole indispensabili per lo sviluppo ed il corretto funzionamento della maggior parte degli organismi viventi. Dal punto di vista chimico, il DNA è un polimero organico a doppia catena i cui monomeri sono chiamati nucleotidi. I nucleotidi sono costituiti da tre componenti fondamentali: un gruppo fosfato, uno zucchero pentoso e una base azotata che si lega al desossiribosio. Le basi azotate che entrano nella formazione dei nucleotidi sono quattro: adenina, timina, citosina e guanina. Il DNA può essere globalmente definito come una doppia catena polinucleotidica (A, T, C, G), antiparallela, orientata, complementare, spiralizzata, informazionale [13]. Le quattro basi sono una componente di studio fondamentale della recente disciplina chiamata Bioinformatica.

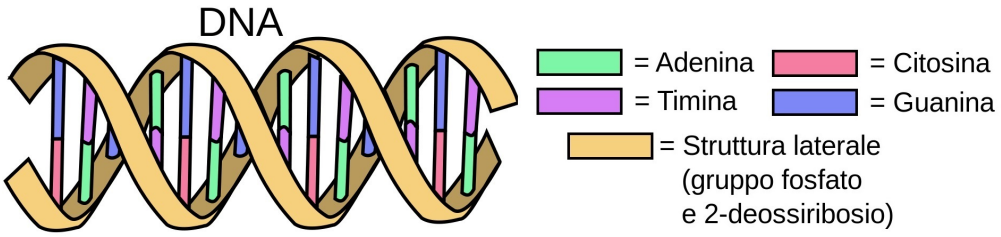

Figura 1.1: Esempio di rappresentazione schematizzata del DNA [13]

1.1.2 Next Generation Sequencing (NGS)

Il Next Generation Sequencing (NGS) comprende una serie di tecnologie di sequenziamento massiccio in parallelo che offrono efficienza, scalabilità e velocità elevatissime. Questo procedimento viene utilizzato per determinare l'ordine dei nucleotidi in interi genomi o regioni mirate di DNA o RNA. L'NGS ha rivoluzionato le scienze biologiche, consentendo ai laboratori di eseguire un'ampia varietà di studi sui sistemi biologici ad un livello mai raggiunto prima. Tutte queste tecnologie hanno in comune una serie di procedimenti rappresentati in figura 1.2.

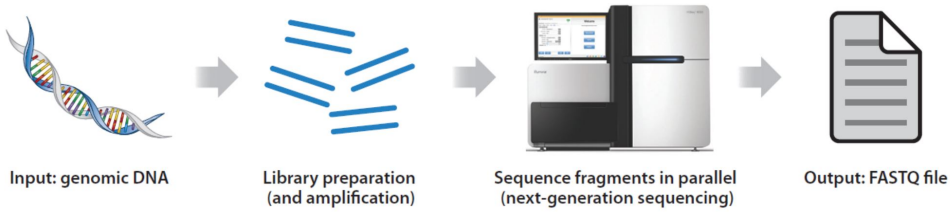

Figura 1.2: Procedimenti NGS[5]

1.1.3 Analisi e digitalizzazione del genoma

Il procedimento per sequenziare e digitalizzare le informazioni rappresentate dal DNA, tramite le tecnologie attuali NGS, si può sintetizzare in quattro passaggi: Frammentazione, Amplificazione, Lettura e Assemblaggio.

- Frammentazione: dopo l'estrazione dell'acido desossiribonucleico avviene la frammentazione o taglio del DNA. La lunghezza ideale dei frammenti dipende dalla tecnologia utilizzata, ad esempio per la tecnologia *Illumina* è di 350-500 bp (*base pair*, coppia di basi) [15].

- Amplificazione: i frammenti di lunghezza fissa vengono replicati decine o centinaia di volte in modo tale da ottenere decine o centinaia di copie dello stesso frammento e tutte della stessa lunghezza.

- Lettura: in questa fase avviene la vera e propria lettura del campione attraverso speciali tecniche che individuano le basi contenute nei campioni in maniera parallela. Queste letture possono avvenire tramite l'utilizzo di fluorescenza o di variazione del PH, che consentono di rilevare la presenza di un

determinato nucleotide. Alla fine di questo processo si ottengono i dati raw[1] sequenziati e memorizzati nel formato standard denominato *fastq*. Questo file, un semplice testo, presenta una formattazione standardizzata rappresentata in figura 1.4. Ogni frammento letto, chiamato read, viene memorizzato in quattro righe:

- header: la prima, chiamata header, inizia con il carattere '@' e serve per memorizzare l'identificativo e alcune informazioni opzionali come il nome dell'esperimento o che strumento è stato utilizzato.
- sequenza: la seconda riga contiene la sequenza nucleotidica della read, questa è la riga più importante e nella quale sono contenute le informazioni principali.
- placeholder: la terza riga invece non contiene quasi mai informazioni, infatti solitamente è composta dal solo carattere '+'. Oltre a questo simbolo, può contenere firme o identificatori.
- quality score: l'ultima riga, infine, contiene gli indici di qualità associate (q) ad ogni base sequenziata. Questi caratteri rappresentano Q che non è altro che la codifica in codice *ASCII* di q definito come:

$$q = -10 \log_{10}(p)$$

dove p è la probabilità che, il nucleotide preso in esame, sia sbagliato. Dovendo q essere rappresentato con un carattere, la codifica di quest'ultimo avviene utilizzando *Phred-33 encoding* che consiste nel sommare, a q, 33 unità, così da saltare i primi 33 caratteri del codice *ASCII*, non facilmente rappresentabili in un file di testo [8].

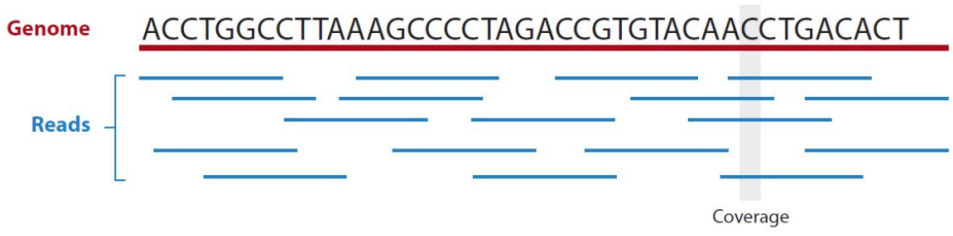

Figura 1.3: Esempio di una singola read, in formato *fastq*[5]

[1]dati privi di un particolare formato di archiviazione

```
@identifier
GGTTTTTTTCACCATCTGTAGACCAAACCAGGAGTAGGCC
+
efffffffcfeffffeefeffdffc']B__Ybbbbbb\]ff
```

Figura 1.4: Esempio di una singola read, in formato *fastq*

- Assemblaggio: come visto dai punti precedenti, i dati ottenuti dai laboratori non consentono di ottenere un filamento DNA consecutivo ma sono tante letture di segmenti del genoma. Partendo dal file in formato fastq, contenente i dati raw, appositi tool molto sofisticati elaborano i dati per assemblare le read, cioè hanno l'obiettivo di memorizzare, in file di tipo fasta[2], delle sequenze più lunghe delle read, queste sono chiamate *contig*[3]. Il procedimento dell'assemblaggio, purtroppo, non riesce a ricostruire l'intera sequenza del DNA ininterrotta, ma comunque produce delle sequenze più lunghe delle read. Questi *contig*, comunque, sono sufficienti per conservare tutte le informazioni del DNA originario.

1.1.4 Problema della conservazione dei dati genetici

I dati dei sequenziamenti prodotti dai laboratori ogni giorno ammontano a centinaia di Tera-byte, questo ha portato, negli ultimi anni, ad uno sviluppo di nuove tecnologie nel campo della compressione di file di testo, per poter memorizzare più efficientemente questi dati. Infatti non esistono soluzioni specifiche per salvare file di testo in maniera compressa, perché non si è mai avuta una quantità tale di dati per cui fosse necessario svilupparle, a differenza di ora. Le compressioni di questi dati, fino ad ora, sono avvenute utilizzando algoritmi general-purpose che implementano algoritmi come LZW, implementato in *Gzip*[4]. Questi algoritmi purtroppo, non essendo sviluppati appositamente a questo scopo, rendono la compressione non ottimale e, trattandosi di banche dati di grandissime dimensioni, non sono sufficienti a risolvere il problema di memorizzazione di questi file. Questo frena la distribuzione e la conservazione dei dati sequenziati è, infatti, a tutti gli effetti un collo di bottiglia per lo sviluppo. A tale scopo negli ultimi anni sono stati sviluppati numerosi tool specializzati in compressione di dati genetici [5] ma

[2]file simile al fastq che a differenza di quest'ultimo contiene solamente le informazione sulla sequenza

[3]Insieme compattato di segmenti di DNA che condividono un overlap tra loro, ad esempio i segmenti ATTG e TTGC formano il contig ATTGC

ad oggi è ancora consuetudine applicare una compressione con tool generalisti per questioni di affidabilità e praticità.

1.2 La Bioinformatica

La bioinformatica è una disciplina scientifica dedicata alla risoluzione di problemi biologici a livello molecolare con metodi informatici. Questa nuova disciplina si è resa necessaria per via dell'enorme mole di dati prodotti dai sequenziatori, dati che possono essere analizzati in maniera efficiente e fornire dei risultati solo da un computer. A questo scopo sono stati sviluppati molti algoritmi per interpretare al meglio i molti nuovi dati a disposizione.

1.2.1 K-mer

Studiando i dati sequenziati che le nuove tecniche producono, si è reso necessario creare nuovi strumenti per assemblare il DNA. A questo scopo sono stati ideati i k-mer. Questi sono sotto-stringhe di lunghezza k contenute all'interno di una sequenza biologica. I k-mer che compongono un intera sequenza di DNA sono chiamati k-mer set.

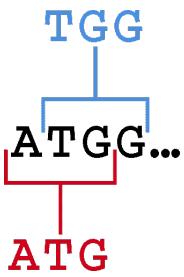

Figura 1.5: Esempio: la sequenza ATGG ha due 3-mer: ATG e TGG. L'insieme {ATG,TGG} è un esempio di *3-mer* set [14].

1.3 Compressione di dati

La compressione dei dati è la tecnica di elaborazione dati che, attuata per mezzo di opportuni algoritmi, permette la riduzione della quantità di bit necessari alla rappresentazione in forma digitale di un'informazione. Questo processo è largamente utilizzato in tutti gli ambiti per memorizzare o trasmettere informazioni. La compressione dati si può dividere in due grandi categorie: Lossy e Lossless.

- **Compressione lossy**: Appartengono a questa categoria tutte quelle tecniche che prevedono algoritmi che, durante la fase di compressione/decompressione, prevedono la perdita di alcune informazioni, spesso sono informazioni non necessarie e non importanti. Questo comporta che dopo la fase di decompressione il file ottenuto non sia uguale a quello originale. Un esempio di algoritmo che utilizza una compressione di tipo lossy e l'algoritmo JPEG.

- **Compressione lossless**: La compressione lossless (senza perdita) prevede il ridimensionamento dello spazio occupato dai dati senza alcuna perdita di informazioni. Spesso questi algoritmi non sono efficienti come gli algoritmi lossy, addirittura in alcuni casi non sono neanche efficaci. Un esempio di programma che utilizza un algoritmo lossless è Gzip che utilizza l'algoritmo LZW. Non avendo quindi perdita di dati, al termine della fase di decompressione, il file ottenuto sarà identico a quello originale

1.3.1 Compressione general purpose o specifica

Gli algoritmi di compressione si possono dividere in ulteriori due categorie: general-purpose o specifici.

- **general-purpose**: gli algoritmi che fanno parte di questa categoria sono detti algoritmi generalisti. Questi non hanno come obiettivo quello di comprimere un solo tipo di file, ma possono operare su tutti i tipi di dati. I programmi di compressione che utilizzano algoritmi general purpose più utilizzati sono sicuramente 7zip e rar, comunemente adoperati dalla maggior parte degli utenti. Questi algoritmi, non conoscendo in anticipo la struttura dei dati su cui operano, non hanno prestazioni ottime.

- **specifici**: questi tipi di algoritmi sono sviluppati per comprimere un solo tipo di file specifico. La conoscenza della struttura dei dati su cui opera, infatti,

dà la possibilità agli sviluppatori di implementare soluzioni più efficienti e puntuali. Nelle presentazioni e analisi successive ci soffermeremo a discutere solo di algoritmi lossless specifici per dati di sequenziamenti genetici.

1.3.2 Elementi utilizzati per la valutazione della compressione

Ci sono diversi indicatori per valutare un algoritmo di compressione e per decidere quale applicare in base al contesto, ad esempio si possono valutare in base alla memoria RAM utilizzata oppure al tempo di calcolo necessario per completare l'operazione. Nel nostro specifico caso utilizzeremo, per la valutazione della compressione, due semplici elementi numerici:compression rate e risparmio di spazio. Definiti come:

- **compression rate**:

$$CR = \frac{Dimensione\ dato\ non\ compresso}{Dimensione\ dato\ compresso}$$

  questo può essere riportato anche sotto forma di rapporto non semplificato. Ad esempio se il dato originale non compresso pesa 200000 byte, il file compresso pesa 40000 byte il Compression Rate risulta di 5, oppure una compressione di 5:1. Perché si possa parlare di compressione il compression rate deve essere >1.

- **risparmio di spazio**:

$$space\ saved = 1 - \frac{Dimensione\ dato\ compresso}{Dimensione\ non\ compresso}$$

  Pertanto, una rappresentazione che comprime la dimensione di archiviazione di un file da 10 MB a 2 MB produce un risparmio di spazio di 1 - 2/10 = 0,8, spesso annotato in percentuale, 80%.

Capitolo 2

K-mer

In questo capitolo vengono trattati i motivi che hanno permesso l'adozione in massa dei k-mer e che li hanno resi così importanti nell'ambito della bioinformatica.

2.1 Creazione K-mer

Come si è visto nel capitolo precedente, i dati ottenuti dal sequenziamento del DNA non consentono di avere una sequenza unica e completa, ma producono solo delle read di lunghezza fissa, senza riferimenti sulla loro posizione e con possibili errori. Negli ultimi tempi i tool di analisi hanno implementato l'uso del corrispondente k-mer set generato, dai dati raw sequenziati, al posto degli stessi dati raw contenuti nei file in formato fastq. Come già detto, i k-mer sono sottostringhe di lunghezza fissa k che vengono estratti dalle reads e utilizzati dai tool. Più precisamente il k-mer set, associato ad una serie di read, è formato da tutte le sottostringhe possibili, di lunghezza k, a partire dalle read (punto C figura 2.2). Ad esempio la read in figura 2.1 viene divisa nel suo 4-mer set composto da tutte le sue sottostringhe di lunghezza 4.

Figura 2.1: Esempio di creazione di 4-mer a partire da una read

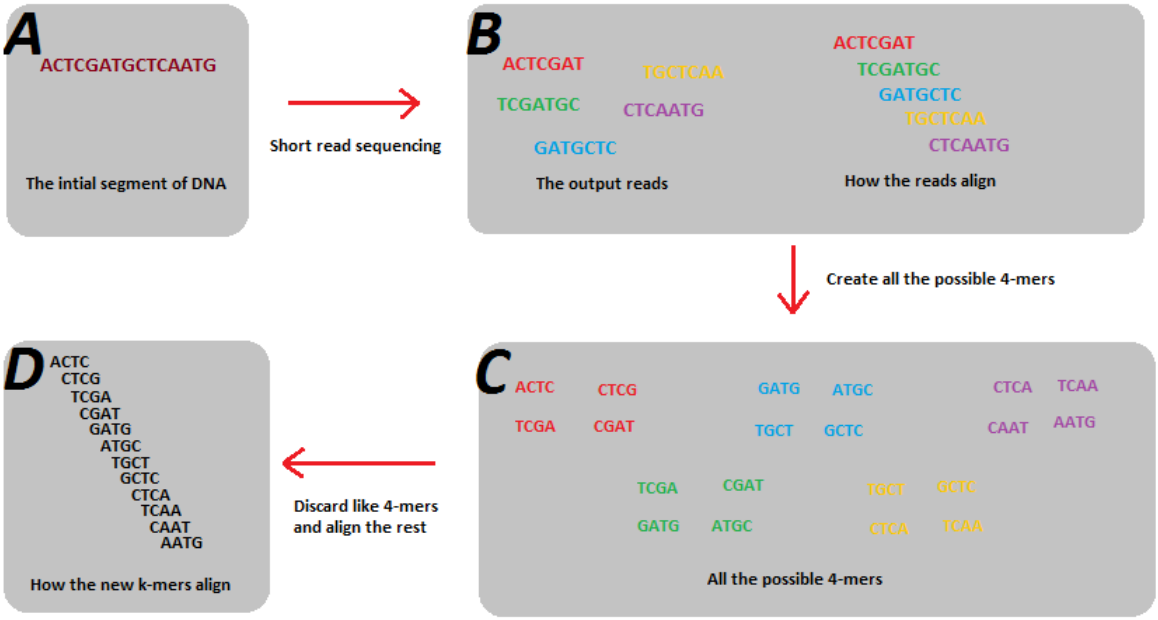

Figura 2.2: Questa figura mostra il processo di scissione delle letture in k-mer più piccoli (4-mer in questo caso) al fine di poter essere utilizzati in un grafico di De Bruijn. (A) Mostra il segmento iniziale di DNA sequenziato. (B) Mostra le letture che sono state ottenute dalla sequenziazione e come si allineano. Il problema dell'allineamento rappresentato in questo punto, però, è il fatto che si sovrappongono per k-2 non k-1 (che è necessario nei grafici di De Bruijn). (C) Mostra che le letture sono divise in 4-mer più piccoli. (D) Scarta i 4-mer ripetuti e quindi mostra il loro allineamento. Si noti che questi k-mer si sovrappongono di k-1 e possono quindi essere utilizzati in un grafico di De Bruijn [14].

2.2 Caratteristiche dei K-mer

Come visto nella sezione precedente la creazione dei K-mer sembrerebbe non utile per comprimere i file fastq. Infatti la creazione dei k-mer aumenta la dimensione di informazioni di questi dati, ad esempio, in figura 2.1, una read di 7 caratteri viene salvata in 16 caratteri (4 caratteri per ogni 4-mer). Il vantaggio dei k-mer, però, è il fatto che questi possiedono delle caratteristiche che possono aiutare a correggere e comprimere i dati raw.

2.2.1 Overlap

Per definizione i k-mer condividono k-1 caratteri con altri k-mer (per esempio, in figura 2.1, il primo 4-mer ACTC ha in comune con il secondo 4-mer (CTCG) le ultime tre basi CTC), infatti, da una read di n basi si ottengono n-k+1 k-

mer e ognuno condivide *k-1* basi con il precedente e con il successivo se non è un end-point[1]. È importante notare che l'operazione di trasformazione di una read in k-mer set sembra non portare benefici in termini di caratteri (In figura 2.1 la sequenza ACTCGAT composta da 7 caratteri genera un 4-mer set con 16 caratteri calcolabili con la formula (n-k+1)*k), infatti, il numero cresce di un fattore k essendo prima sufficienti solo n caratteri. Il k-mer set, generato dalle read di lunghezza totale n, ora necessita di O(n*k) caratteri per essere memorizzato. Il peggioramento che questa operazione comporta è evidente quindi, a causa dello spazio aggiuntivo richiesto, necessitiamo di tecniche per limitare l'inconveniente.

2.2.2 Conteggio dei k-mer

Il DNA è rappresentato da una stringa composta solo da quattro caratteri che si alternano: A,G,C,T. Sapendo che la catena di nucleotidi è molto lunga, rispetto al numero dei caratteri che la compongono e che durante il processo di sequenziamento avviene la fase di amplificazione, (dove vengono replicati decine o centinaia di volte i frammenti ricavati dal campione vedi sec. 1.1.3) è molto probabile che un k-mer (che di solito contiene un numero di basi molto inferiore rispetto alle read lette dal campione sequenziato) presente in una read si ripresenti anche in altre read sequenziate. Per questo motivo molti tool hanno l'obiettivo di contare quante volte un singolo k-mer è presente in un k-mer set. Come vedremo tutte e due i tool presentati, avranno come aggiunta un file con all'interno i conteggi dei k-mer

2.3 Vantaggi dell'utilizzo dei K-mer

2.3.1 Errori e repeat

Errori Come abbiamo appena visto, i k-mer hanno la possibilità di essere contati. Questi conteggi, sebbene non facciano parte del DNA, evidenziano le porzioni più frequenti e quindi portano con sé numerose informazioni importanti per un DNA sequenziato ma non assemblato. Per questo motivo è necessario ricavare anche loro durante la fase di creazione del k-mer set e tenerne traccia. Grazie a questi conteggi infatti si ha la possibilità di individuare quali k-mer presentano

[1]k-mer ottenuto dall'ultima sottostringa di una read

un errore di sequenziamento. Abbiamo visto che essendo il DNA presente molte volte nelle stringa sequenziata, anche se in frammenti disordinati, è improbabile che ci sia un k-mer che sia presente raramente, questo si traduce in un aiuto per la ricerca di errori tramite l'analisi dei conteggi. È per questo motivo quindi che, un k-mer con molteplicità inferiore ad una certa soglia, viene scartato perché con alta probabilità contiene almeno un errore di sequenziamento, quasi impossibile da correggere.

Repeat I conteggi non sono importanti solo per individuare gli errori di sequenziamento, ma tornano utili anche nelle analisi, infatti un k-mer molto frequente all'interno di un DNA può indicare ad esempio la presenza di un repeat[2].

2.4 Rappresentazione

Per memorizzare questi dati sono stati sviluppati svariati metodi che si possono dividere in due categorie: general purpose e specializzati (basati su grafi di overlap).

- Rappresentazioni general purpose: Nei computer, per memorizzare dei dati, si utilizzano degli oggetti astratti chiamati strutture dati, questi permettono di organizzarli in maniera coerente e renderli accessibili nel modo più efficiente possibile. Un k-mer set è composto da un insieme di stringhe di lunghezza fissa k, quindi per essere memorizzato si possono utilizzare le classiche strutture dati usate per salvare i caratteri come array, liste, linked list, set e grafi. Queste rappresentazioni, però, non tengono conto degli overlap tra k-mer contigui, ciò significa che vengono memorizzate moltissime informazioni (basi) ridondanti.
- Rappresentazioni basate su grafi di overlap: per rappresentare questi dati, lo strumento che più è adatto, è proprio il grafo di overlap, in particolare quello di de Bruijn (figura 2.3). Un grafo di de Bruijn è un grafo direzionale nel quale ogni nodo rappresenta un k-mer ed ogni vertice rappresenta un overlap tra due k-mer (è direzionale perché i vertici hanno un solo verso di

[2]Il DNA di molte specie è caratterizzato da una alta ripetitività nella disposizione delle basi, queste sottostringhe chiamate repeat si ripetono più volte lungo il DNA e rappresentano un problema molto importante perché creano ambiguità durante la fase di allineamento e assemblaggio del DNA[16].

percorrenza, ad esempio, il vertice v tra due nodi n_1 e n_2 indica che il suffisso, di k-1 basi, del k-mer associato ad n_1, corrisponde al prefisso, di k-1 basi, del k-mer associato al nodo n_2). Un k-mer set può essere rappresentato in maniera univoca con un grafo di De Bruijn e viceversa.

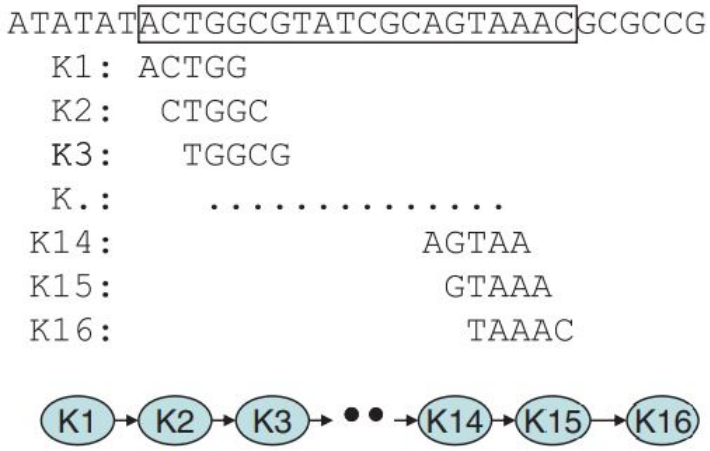

Figura 2.3: In questo esempio le read sono state divise in k-mer ($K = 5$ bp). Ci sono in totale 16 diversi k-mer, molti dei quali derivano da più read. I k-mer sono disposti in modo ordinato lungo il genoma in base alla loro posizione iniziale e alla struttura nel Grafico DBG. Questo è illustrato nella parte bassa. Si noti che la maggior parte dei nodi hanno un solo arco entrante e un solo arco uscente [9]

.

Rappresentazione su disco La rappresentazione tramite grafo di De Bruijn è conveniente per le analisi del genoma ma, in termini di spazio su disco, non porta benefici rispetto alle strutture dati più classiche citate sopra. Questo è dovuto dal numero di k-mer, che rimane invariato, e da fatto che ogni k-mer viene memorizzato sempre con k caratteri. Da questo grafo però si evidenzia un potenziale miglioramento attuabile tramite l'eliminazione degli overlap che, per definizione, sono informazioni ridondanti. Questo è un primo esempio di tecnica per un compressore specializzato per sequenze genetiche.

Capitolo 3

Presentazione Tool

In questo capitolo verranno presentati i due tool studiati per la compressione, con conteggi, di sequenziamenti genetici. Sebbene tutti e due questi tool abbiano la possibilità di operare anche in altri modi, nelle prossime presentazioni ed analisi verranno considerati solo gli aspetti di compressione e conteggio.

3.1 UST

Questo tool, sviluppato da Rahman e Medvedev nel 2020 [11] ha lo scopo di comprimere i dati ottenuti da un sequenziamento partendo da un grafo di de Bruijn compattato. Prima di procedere con la presentazione, però, vedremo una breve descrizione del software che, partendo dai dati raw, produce questo grafo compattato: Bcalm.

3.1.1 Bcalm

Questo tool, sviluppato da Chikhi et al. nel 2016 [3] è uno strumento molto efficiente per completare l'assemblaggio, fase più costosa e complicata di tutta la pipeline che i dati genetici affrontano. L'innovazione più importante che introduce questo strumento, è la compattazione del grafo di de Bruijn attraverso l'uso degli unitig. Bcalm, infatti, partendo dai dati raw, ottiene un grafo di de Bruijn compattato. Un grafo di de Bruijn compattato, a differenza di uno classico, non contiene tutti gli overlap nella sua rappresentazione. Esso infatti forma tutti gli unitig e non salva i k-mer che li compongono. Questa procedura porta già notevoli miglioramenti in fatto di memorizzazione. Bcalm, quindi, genera il k-mer set ed, inoltre, si occupa di costruire il grafo di de Bruijn compattato. Oltre a ciò Bcalm

ha la possibilità di generare anche i k-mer counting (conteggi) e di eliminare i k-mer con frequenza bassa. È da segnalare che, solitamente, l'operazione di compattazione risulta essere molto costosa in termini di tempo, per questo il tool è stato ottimizzato per velocizzare le operazioni. Bcalm, infatti, usa speciali strutture dati ed algoritmi paralleli. In output restituisce un file in formato fasta dove si alternano due tipi di righe (come si vede in figura 3.1) la prima contiene l'header mentre la seconda la sequenza di basi che rappresentano l'unitig compattato dal tool.

```
>0 LN:i:38 ab:Z:8 8 8 8 8 8 8 8   L:-:1043129:+
AAATCTTCTACTAGTTTTTTCAGTATTCTACAGTTGTT
>1 LN:i:31 ab:Z:9    L:-:978260:+
CATCAGCAGTCAAAAAAATAAAAATCGCGTC
>2 LN:i:35 ab:Z:18 18 18 18 18   L:-:745024:+ L:-:746118:+  L:+:977560:+
CGGTACAAAAACGCGTACACACACACAGGGCGCGG
>3 LN:i:31 ab:Z:2   L:-:1170355:+
GAGGAGACGGATGGTGTAGAAAATAGCCGGC
```

Figura 3.1: Alcune righe di esempio in output del tool Bcalm2 utilizzando, come input, la sequenza di ingresso SRR13605073. Nella prima riga dell'header, preceduto dal simbolo >, troviamo un id di riferimento, la lunghezza dell'unitig, i conteggi dei k-mer contenuti nell'unitig ed infine eventuali archi utili per la costruzione del grafo di de Bruijn compattato. Nella seconda riga troviamo invece la sequenza di basi che rappresentano l'unitig

3.1.2 Presentazione UST

UST(Unitig Stitch) è un tool sviluppato da Rahman e Medvedev nel 2020 [11]. Questo programma prende in ingresso il grafo di de Bruijn compattato (nel nostro caso il file fasta restituito da bcalm, figura 3.1) e va a compattare ulteriormente la rappresentazione del k-mer set rispetto a Bcalm. Tutto ciò avviene senza perdita di informazioni grazie alla proprietà SSPS[1] introdotta dagli stessi autori e da Břinda et al. [1] con il nome di Simplitig. L'algoritmo implementato in UST esplora una e una sola volta tutti i vertici del grafo compattato, ricevuto in input, e ad ogni ramificazione sceglie in maniera arbitraria il nodo da fondere con quello in esame. Questo genera un unitig di lunghezza maggiore che d'ora in poi chiameremo simplitig (esempio di funzionamento in figura 3.4). Questo tool non ha

[1]Spectrum-Preserving String Set, proprietà che indica la possibilità di rappresentare un k-mer set in maniera univoca tramite l'uso di sequenze compattate. Queste sequenze raggruppano i k-mer che condividono un overlap tra loro.

il solo obiettivo di creare unitig massimali o simplitig molto lunghi ma anche di creare il minor numero possibile di queste sequenze. Questo perché se ottenessimo pochi unitig molto lunghi e moltissimi unitig di lunghezza k, cioè della stessa lunghezza dei k-mer originari, impediremmo un vero miglioramento dello spazio di archiviazione. Infatti, se il numero di righe da memorizzare su file fosse molto simile al file in ingresso, non otterremmo dei file con dimensione sul disco minore. L'innovazione di questo tool consiste proprio nel suo algoritmo greedy[2] per creare i simplitig, infatti è molto veloce ma allo stesso tempo si avvicina molto al lower bound teorico calcolato dagli stessi autori [11].

Output di UST UST produce più file in output, se opportunamente specificato. Questo tool, infatti, oltre a creare la struttura dati del grafo con i simplitig, produce un file con i conteggi di queste sequenze. I file che produce hanno come estensioni *.ust.fa* e *.ust.counts*. Il primo file, come si vede in figura 3.2, contiene la rappresentazione SPSS dei k-mer sotto forma di file di testo. Come si può vedere questi simplitig, a differenza delle read e dei k-mer, non hanno tutte la stessa lunghezza. Il secondo file, invece, con l'estensione .ust.counts, contiene tutte i conteggi dei simplitig. Questi conteggi sono salvati come semplice vettore di interi, in particolare gli interi sono scritti uno per riga, come si vede da figura 3.3.

Ulteriore compressione Come specificato nella pubblicazione che presenta il tool UST, [11] questi file di output devono passare per un ulteriore miglioramento. La rappresentazione del k-mer set in due file separati viene eseguita anche per poter comprimere in maniera separata, e con due tipi diversi di algoritmi di compressione, questi due output. Nelle prossime analisi in particolare utilizzeremo 7zip, che implementa LZMA, sia per il file fasta che per il file dei conteggi (file di testo che contiene solo numeri interi positivi).

[2]Un algoritmo greedy è un algoritmo che cerca una soluzione ammissibile da un punto di vista globale attraverso la scelta della soluzione più appetibile per quel determinato programma a ogni passo locale.

```
>
CAGCTGCGGCTCCAGGGACAAAAACGACGCTGGCGCCCCTGCGCCTGCCGCCGCCGCCGCCATGCTGTTC
>
TGCGGCTCCAGGGACAAAAACGACGCTGGCACCCCCGCGCCTGCCGCCGCCGCCGCCATGCTGTTCCACTCGCCGCGACCGCC
>
GACGCGATTTTTATTTTTTTGACTGCTGATGCTGTGCGTTGCGTCGGTCTCATCCCCTCTGTCATCCTTCGTAGAAACTTGCGGTGTGCTG
```

Figura 3.2: File output con estensione *.ust.fa* del tool UST, utilizzando come input la sequenza SRR13605073.

```
13
8
8
8
5
```

Figura 3.3: File output con estensione *.ust.counts* del tool UST, utilizzando come input la sequenza SRR13605073.

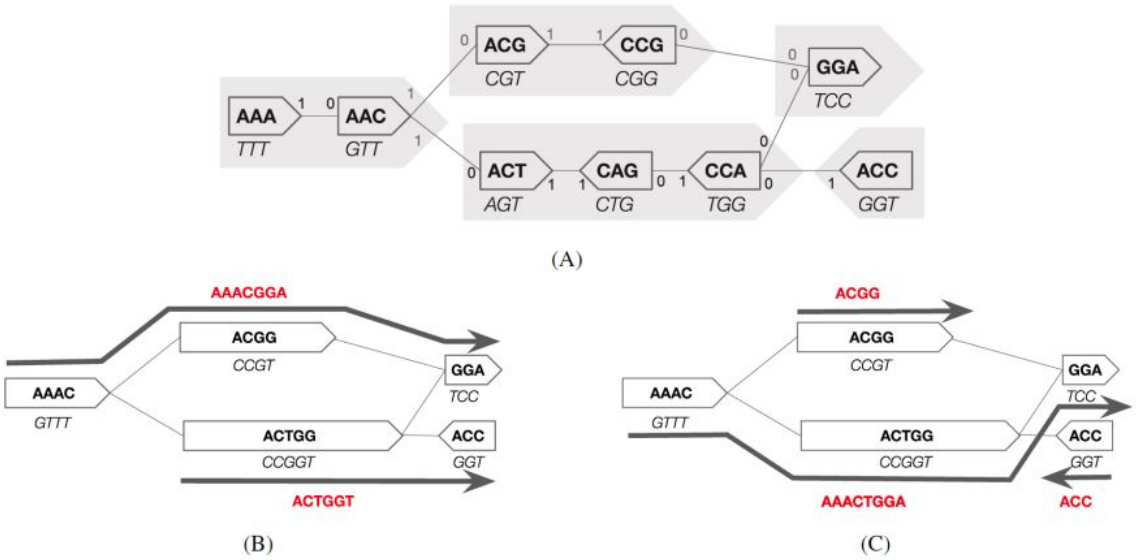

Figura 3.4: Esempio di funzionamento del tool UST: in A abbiamo il grafo di de Bruijn di un 3-mer set, in B e C abbiamo il grafo compattato restituito da bcalm ed infine in rosso abbiamo i Simplitig prodotti dall'esecuzione di UST. Le frecce grige sono dei percorsi che l'algoritmo greedy di UST potrebbe scegliere. Come si vede sia nella figura B che C sono stati creati dei simplitig, ma in quella B sono stati creati in modo più efficiente [11].

3.1.3 SPSS

Gli autori del tool UST, nella pubblicazione [11] introducono il concetto di proprietà SPSS. Questa proprietà fa si che un set di k-mer possa essere rappresentato, in maniera univoca, dalla fusione delle sequenze che raggruppano i k-mer che condividono un overlap tra loro. Come visto sopra questa operazione elimina molti dati ridondanti ma conserva tutta l'informazione contenuta nel k-mer set. Questa operazione non riguarda l'assemblaggio effettuato ad esempio da bcalm ma la sua applicazione si focalizza esclusivamente sulla compattazione (con risultati molto vicini al lower bound teorico) degli unitig prodotti da bcalm. Questo crea quindi delle lunghe sequenze chimeriche[3] non appartenenti al DNA in esame.

[3]Con il termine chimeriche si identificano quelle sequenze che vengono prodotte dalle analisi sulle read sequenziate ma che non sono reali sottostringhe di DNA.

3.2 Counting de Bruijn Graph

In questa sezione andremo a presentare una nuova rappresentazione, chiamata Counting de Bruijn Graph, introdotta da M Karasikov et al. [6] che è implementato nel framework Metagraph [7] sviluppato dagli stessi autori. Sebbene il solo grafico di de Bruijn contenga le informazioni necessarie per studiare l'intero genoma, non è efficiente per le fasi di analisi quantitative delle read. In particolare non è possibile rispondere alle domande relative alla posizione esatta della presenza di una sequenza ed alla relativa frequenza con cui questa è presente nel genoma. Questo problema si può affrontare aggiungendo delle annotazioni per rappresentare le relazioni tra k-mer e i file di input. Come abbiamo visto nel tool precedente questi dati aggiuntivi, i conteggi nell'esempio di UST, vengono salvati in precise strutture dati, un vettore. Questo nuovo approccio ha lo scopo di rappresentare in maniera efficiente questi dati aggiuntivi, chiamati *label* (etichette). In questo approccio, oltre ai conteggi, vengono introdotte delle *coordinate dei k-mer* (*k-mer coordinates* [6]). In queste analisi e presentazioni non verrà valutata la presenza di queste coordinate.

3.2.1 Presentazione rappresentazione Counting de Bruijn Graph

Il problema di rappresentare gli attributi (conteggi nel nostro caso) viene gestito da questa rappresentazione tramite l'utilizzo di due diversi elementi: il grafo di de Bruijn e le *annotazioni estese*(*extended graph annotations*). Questi due elementi, infatti, costituiscono la struttura dati astratta chiamata *Countnig de Bruijn graph*. Queste annotazioni devono essere gestite per poter essere rappresentate sul disco in maniere efficiente. Infine questa nuova rappresentazione si occupa anche di interrogare questi dati senza il bisogno di una completa decompressione, così da risultare efficiente anche nel momento di utilizzo di dati e non solo nella fase di salvataggio. Tutte le annotazioni aggiuntive rispetto al grafo di de Bruijn vengono rappresentate come matrici (vettori nel caso di matrici monodimensionali, come i conteggi). Queste matrici o vettori, però, presentano molti campi nulli, sono quindi chiamate matrici sparse. Per rappresentarle in modo efficiente vengono create altre due strutture dati: una matrice binaria ed un vettore contenente i valori numerici delle celle.

- La matrice binaria: è una matrice che contiene solo dati binari ({0,1}) ed ha il compito di rappresentare le corrispondenze dei valori con i k-mer. Questa matrice, inoltre, definita come *sparsa*, ha molti dei suoi valori che valgono zeri(figura 3.5 pannello A matrice B).

- Il vettore di valori: questo vettore, ha il compito di contenere tutti i valori presenti nella matrice di partenza, quelli delle *label*, sotto forma di array (figura 3.5 pannello A matrice A).

Questa rappresentazione sdoppiata delle annotazioni dà la possibilità di comprimere in modo efficiente i dati al proprio interno. Nel nostro caso di studio un esempio di rappresentazione dei conteggi è mostrato in figura 3.5 pannello B. In questo esempio possiamo distinguere la matrice monodimensionale, quindi vettore, B e l'array A contente i valori dei conteggi.

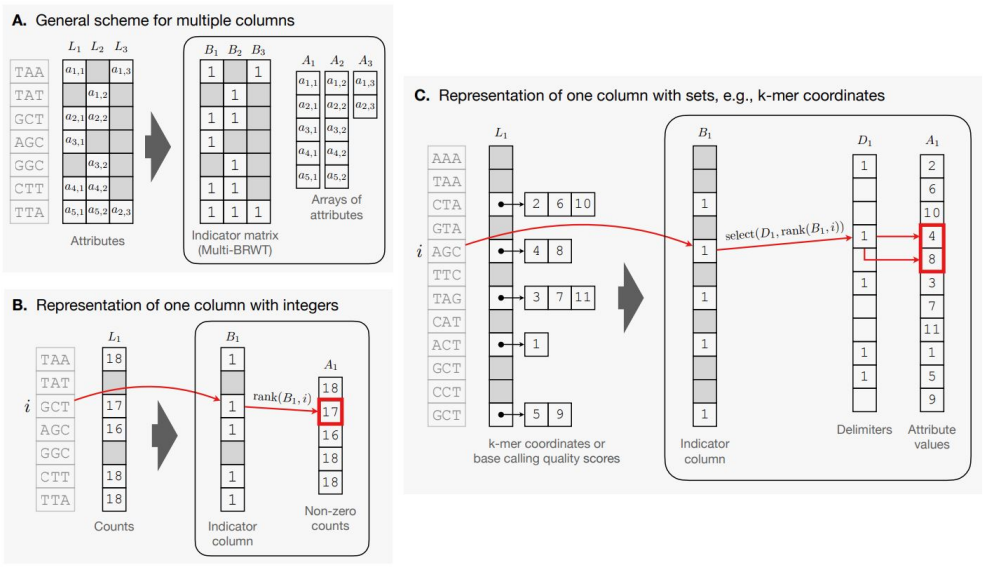

Figura 3.5: La rappresentazione delle matrici sparse in forma compressa. **Pannello A**: Schema generale per matrici sparse con attributi astratti, dove gli attributi non assegnati vengono eliminati da un indicatore matrice binaria archiviata in una rappresentazione compressa. Essa supporta l'operazione di classificazione sulle sue colonne per consentire l'accesso all'attributo corrispondente per una data cella della matrice. Gli attributi vengono archiviati separatamente, in genere sotto forma di array compressi. **Pannello B**: Lo schema applicato a singola colonna con valori interi (ad es. i conteggi dei k-mer) e l'algoritmo di query (richiesta) (ad es. il conteggio di k-mer i è recuperato come *A1[rank(B1, i)]*). Le celle vuote in grigio rappresentano gli zeri. **Pannello C**: Lo schema applicato ad una singola colonna in cui ogni cella è un insieme di numeri o una tupla (ad esempio, che rappresenta le coordinate k-mer) [6].

3.2.2 Implementazione

Questo nuovo approccio è stato implementato dagli autori utilizzando il framework Metagraph. Questo framework consente di creare il grafo di de Bruijn, eventualmente pulirlo, ed annotarlo. A differenza del tool precedente, quindi, questo nuovo approccio non si concentra nel rielaborare i dati ottenuti da Bcalm, ma produce e comprime i dati autonomamente.

Compressione Come appena visto questo tool assembla e produce il grafo di de Bruijn e i conteggi. Oltre a ciò, Metagraph si occupa della compressione di questi output. In particolare per comprimere i conteggi viene applicata una codifica delta. Essendo infatti i conteggi una funzione di aggregazione di percorsi contigui, questo porta che cambino in modo incrementale. Tale proprietà fa si che un conteggio possa essere approssimato con la media dei conteggi adiacenti e possa essere compresso con una codifica delta. Dopo aver applicato la codifica delta, la matrice dei conteggi viene scomposta nei suoi due elementi: la matrice binaria(con le stesse dimensioni) e l'array con i valori. La matrice binaria viene salvata nel formato di compressione Multi-BRWT mentre l'array di valori viene salvato separatamente. L'array, infine, dopo una manipolazione che rende tutti i valori positivi, viene ulteriormente compressa usando l'algoritmo implementato in Directly Addressable Codes(dac_vector) [2] (come si vede in figura 3.6).

Output Come precedentemente accennato, questo tool produce autonomamente il grafo di de Bruijn. Questo è salvato da Metagraph in un file con l'estensione *.dbg*. Questo file, a differenza del tool precedente non è un semplice file di testo, ma è già compresso e quindi illeggibile autonomamente. È infatti il tool a mettere a disposizione strumenti per la lettura. Successivamente, questo tool dà la possibilità di pulire il grafo, e di annotarlo. L'operazione di annotazione, produce due file *annotation.column.annodbg* ed *annotation.column.annodbg.counts* che contengono i conteggi. Questi output sono già compressi, ma comunque leggibili tramite l'invocazione di speciali comandi forniti da Metagraph. È importante sottolineare che, gestendo già autonomamente la compressione, l'utilizzo di Metagraph risulta molto più snello dei tool precedenti.

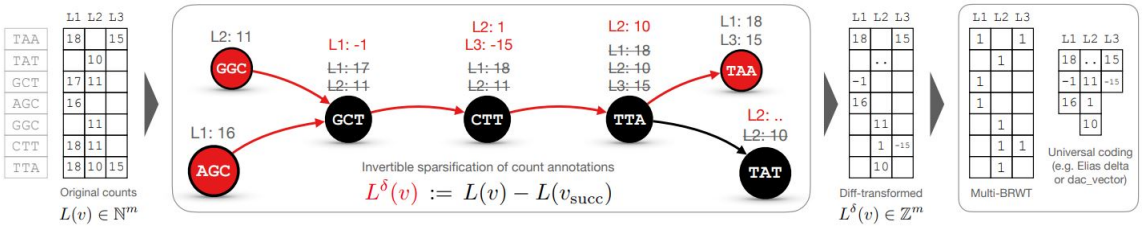

Figura 3.6: Un diagramma schematico che illustra la codifica dei conteggi k-mer in m colonne con l'approccio proposto dalla pubblicazione [6]

Capitolo 4

Analisi

4.1 Dataset utilizzati e Parametri utilizzati

4.1.1 Dataset utilizzati

Per le prossime analisi, abbiamo selezionato come dataset le sequenze riportate in tabella 4.1. I dataset proposti sono una selezione tra quelli utilizzati in entrambe le pubblicazioni dei tool esposti nel capitolo precedente, [11] e [6], con l'aggiunta di qualche nuovo dataset scelto arbitrariamente. In questo modo si evitano eventuali bias o favoritismi per uno specifico tool e si aggiunge della variabilità nel tipo di dato inserito. Avere un dataset ampio e variabile è importante perché la distribuzione dei dati genetici varia molto con il tipo di specie e di tecnica utilizzata per sequenziare. Per questo motivo è stato ritenuto importante utilizzare in queste analisi un dataset reale, con diversi tipi di specie e diverse tecniche di sequenziamento. In generale questi dataset sono stati scelti anche per le loro dimensioni, che dovevano essere ridotte, avendo a disposizione computer con potenze di calcolo limitate. Nella tabella 4.1, sono riportate le informazioni base dei campioni di sequenziamento utilizzati. Questi sono stati recuperati tutti da NCBI SRA [12], banca dati americana. I dati di questa tabella sono stati presi dalle pagine di riepilogo dei vari campioni.

Dataset	**Denominazione SRA**	**Lunghezza read**	**# di read**
Musa balbisiana; RNA-Seq	SRR10260779	101	22,113,556
SARS-CoV-2	SRR21356753	101	31,753,350
Human gut metagenome	SRR341725	90	12,739,564
Drosophila ananassae	SRR332538	75	9,182,963
Broiler Chickens	SRR13605073	91	7,381,614
Homo sapiens; RNA-Seq	SRR957915	101	24,729,920
Subway and urban air samples	SRR10002688	151	1,191,556
Soybean RNA-seq	SRR11458718	125	41,307,254

Tabella 4.1: Caratteristiche del dataset utilizzato nelle analisi.

4.1.2 Parametri utilizzati

I due tool sotto esame hanno lo scopo di comprimere solo la sequenza genetica, non gli header o altre informazioni presenti nei file fasta. Per questo motivo tutti i file, scaricati tramite il portale NCBI [12], sono stati processati rimuovendo tutte le informazioni non gestite dai tool. Questa operazione ha portato l'intero dataset in esame a pesare 29,6 Gb contro i 44.4 Gb iniziali (vedi tabella 4.1). Per le prossime analisi si è inoltre utilizzata una lunghezza dei k-mer uguale a 31 (k = 31). Infine, si è deciso di non eliminare nessun k-mer con molteplicità bassa, questo perché non abbiamo modo di confrontare l'efficacia di questa procedura. Sebbene questi non siano approcci ottimali per un eventuale utilizzo pratico, ci permettono di avere una confronto più uniforme tra i due tool. Queste due scelte, infatti, sono state prese per cercare di comparare al meglio i due tool, non concentrandosi su un risultato valido per un eventuale utilizzo reale. Come precedentemente discusso, a tutti e due questi tool è stato inserito come opzione anche quella dei conteggi dei K-mer.

```
@
CATCTTCCCCGAGGCCCAGCAGGGCAGCAGCATGCTCTTTTCGCCCGTGCCGCCCGCCACGATGATGGGCATGGAATCGAAAAACGGCTT
@
GGCATGGCTGCGCGGACACGACCGGCCCTACCGCCGGAGCATCATTCTGCTCGTCGCGCCGGACACCGCGCAGATGCTCGCAAGGGAAGG
@
ACCTTGCGTTATTTGGCTGAGGGAAATGACCATATGACTGCTTTTGCCAAAGTGGGCGATGCGGATAAATTGCTGCCTGAAAAGGCCACA
@
CCGGATTCGGATCAGTAATGGGGATAGGTTCCACACGGCTGTGGGTATCATTAGTATGAAGTATCAAAAGTTCTTTGATATCCTGTGCTG
```

Figura 4.1: File in input fasta senza l'header. In particolare della sequenza SRR341725

Denominazione SRA	**.FASTA**	**No header .FASTA**	**SRA fasta.gz**
SRR10260779	7916540350	4599619648	1819339768
SRR21356753	9199662567	6580027064	2328811237
SRR341725	3876957494	2369558904	958395338
SRR332538	2462362870	1432542228	394451945
SRR13605073	979724181	694063443	216997431
SRR957915	8755969818	5143823360	2011767643
SRR10002688	457879312	366999248	121753808
SRR11458718	14000021948	10574657024	3391637315
Totale	*47649118540*	*31761290919*	*11243154485*

Tabella 4.2: Spazio su disco utilizzato dai vari formati dello stesso file. Nella prima colonna troviamo gli identificativi dei dataset, nella seconda troviamo, in Bytes, la dimensione del file fasta non compresso. Nella terza colonna, invece, troviamo lo spazio occupato dallo stesso file ma con gli header omessi. Infine nell'ultima colonna troviamo la dimensione del file originariamente scaricato dal portale. L'unità di misura utilizzata è sempre il Byte.

4.2 Dati ottenuti

In questa ultima sezione verranno presentati i risultati ottenuti dopo l'utilizzo del tool UST e del metodo *Counting de Bruijn Graph* implementato in Metagraph.

4.2.1 Dati di 7zip e Gzip

Per comparare i due tool, UST e Metagraph, anche a livello globale, è stato deciso di riportare i dati di compressione di 7zip e Gzip, programmi di compressione maggiormente utilizzati, per i set di dati in esame. Verranno riportati solo pochi dati sintetici perché i due programmi non sono oggetto principale di questo studio.

Denom. SRA	**No h. .FASTA**	**R. Gzip**	**R. 7zip**	**S.S. Gzip**	**S.S. 7zip**
SRR10260779	4599619648	3,40	5,80	70,569	82,750
SRR21356753	6580027064	3,55	5,85	71,830	82,908
SRR341725	2369558904	3,35	4,13	70,176	75,795
SRR332538	1432542228	5,91	13,17	83,087	92,404
SRR13605073	694063443	4,26	6,41	76,522	84,409
SRR957915	5143823360	3,42	5,61	70,718	82,164
SRR10002688	366999248	3,48	4,64	71,249	78,431
SRR11458718	10574657024	3,79	8,19	73,618	87,788
Totale	31761290919	3,64	6,36	72,543	84,268

Tabella 4.3: Spazio su disco utilizzato dai vari formati dello stesso file. Nella prima colonna troviamo gli identificativi dei dataset, nella seconda troviamo, in Bytes, la dimensione del file fasta non compresso con gli header omessi. Nella terza e quarta colonna, invece, i compression rate dei due programmi. Infine nelle ultime due colonne troviamo il risparmio di memoria (*saved space* in percentuale)

4.2.2 Risultati Ust

In questa sezione verranno riportati i risultati ottenuti dopo la compressione dei campioni, con conteggi, del tool UST. I dati ottenuti dopo l'utilizzo di UST sono stati compressi, come indicato nella pubblicazione [11], con *7zip*.

Denominazione SRA	**no header .FASTA**	**UST(.7z)**	**C.R. ratio**
SRR10260779	4599619648	106759875	43,08
SRR21356753	6580027064	368111814	17,88
SRR341725	2369558904	127882291	18,53
SRR332538	1432542228	19342634	74,06
SR13605073	694063443	27559461	25,18
SRR957915	5143823360	254117717	20,24
SRR10002688	366999248	49039023	7,48
SRR11458718	10574657024	106169150	99,60
Totale	*31761290919*	*1058981965*	**29,99**

Tabella 4.4: Risultati ottenuti dalla compressione di UST. Nelle colonne centrali sono riportate, in Bytes, le dimensioni dei file in esame, prima e dopo la compressione. Nell'utlima colonna è riportato il compression rate.

Osservazioni

Come si vede dalla tabella 4.4, i risultati della compressione sono abbastanza omogenei e il valore totale del compression rate risulta di **29.99**. Questo risulta molto elevato se confrontato con i risultati ottenuti tramite algoritmi general purpose di 7zip e Gzip. I file di output di UST sono stati entrambi compressi con 7zip. Il motivo di questa scelta è dovuto alla miglior compressione che quest'ultimo programma offre rispetto ad altri algoritmi come MFCompressor [10]. Dalle poche prove fatte infatti 7zip risultava il miglior compressore.

4.2.3 CountingBDG

I risultati ottenuti utilizzando il framework Metagraph [7] sono riportati nella tabella 4.5. Come già presentato, questo tool ottiene già dei dati compressi che, quindi, non devono essere successivamente compressi con programmi general purpose, a differenza di UST. Questo perché se si comprimessero non sarebbero più invocabili dallo stesso Metagraph per l'analisi.

Denominazione SRA	**no header .FASTA**	**ContingDBG**	**Compression ratio**
SRR10260779	4599619648	511531648	8,99
SRR21356753	6580027064	1884763752	3,49
SRR341725	2369558904	678940469	3,49
SRR332538	1432542228	96243052	14,88
SRR13605073	694063443	124412268	5,58
SRR957915	5143823360	1138587469	4,52
SRR10002688	366999248	280662927	1,31
SRR11458718	10574657024	519665575	20,35
Totale	***31761290919***	***5234807160***	***6,07***

Tabella 4.5: Risultati ottenuti dalla compressione di CounitgDBG. Nelle colonne centrali sono riportate, in Bytes, le dimensioni dei file in esame, prima e dopo la compressione. Nell'utlima colonna è riportato il compression rate

Osservazioni

Come si vede dalla tabella 4.5 i risultati non sono troppo omogenei il compression rate totale è di solamente **6.07**, addirittura minore del compression rate ottenuto con 7zip. Questi due valori però non sono realmente confrontabili, perché i file ottenuti da Metagraph non contengono solamente la sequenze in input ma comprendono anche i conteggi. Questo è anche il motivo per cui il risultato ottenuto dalle nostre prove è ben distante dal compression rate di 14 riportato nella pubblicazione [6].

4.2.4 Risultati a confronto

Presentiamo quindi alcuni dati per mettere a confronto i due tool.

Denom. SRA	S.s. UST	S.s. CountingDBG	S.s. Ust/CountingDBG
SRR10260779	97,679	88,879	79,13
SRR21356753	94,406	71,356	80,47
SRR341725	94,603	71,347	81,16
SRR332538	98,650	93,282	79,90
SRR13605073	96,029	82,075	77,85
SRR957915	95,060	77,865	77,68
SRR10002688	86,638	23,525	82,53
SRR11458718	98,996	95,086	79,57
Totale	96,666%	83,518%	**79,77%**

Tabella 4.6: Riepilogo dei due tool tramite il *saved sapce*. Nelle colonne centrali sono riportati i valori di risparmio di spazio, in forma percentuale, rispetto ai dati *.fasta* senza header. Nell'ultima colonna è riportato lo spazio risparmiato da UST nei confronti di CountingDBG

Osservazioni

Come dall'ultima tabella (4.6) tutti e due i tool presentano un miglioramento di spazio notevole rispetto al file non compresso. Inoltre, grazie all'ultima colonna, si può valutare come UST abbia compresso, in maniera più efficiente, i dati iniziali addirittura con un risparmio di spazio di **79,77%** minore rispetto al metodo CountingDBG.

4.2.5 Osservazioni Finali

Come visto i dati ottenuti ci hanno dato la possibilità di osservare le dimensioni dei dati compressi che questi due tool producono. Grazie all'utilizzo di parametri come il compression rate e il risparmio di spazio, abbiamo potuto confrontare i risultati di questi due tool. I compression rate che abbiamo ottenuto sono tutti maggiori di 1, quindi i due tool hanno sempre compresso efficacemente i dati in ingresso. Osservando la tabella conclusiva 4.7, possiamo dire che UST rappresenta in maniera più efficiente i dataset, cioè comprime maggiormente i dati. Grazie all'ultima colonna si vede che Metagraph, tramite la rappresentazione Countig de Bruijn Graph, produce dei file che sono **4.95** volte più grandi rispetto a quelli di UST. Infine, il risparmio di memoria che si ottiene utilizzando UST al posto di Metagprah, compreso il salvataggio dei conteggi, è di circa dell'80%.

	UST	**Contig Dbg**	**UST rispetto a Coutig DBG**
Compression Rate	**29,99**	6,07	4,94
Saved Space	**96,666**	83,518	79,77

Tabella 4.7: Rappresentazione dei dati finali. In grassetto vengono segnati i dati ottenuti che portano maggior miglioramento dal punto di vista della compressione. L'ultima colonna rappresenta dei dati aggiuntivi per la comparazione dei due tool

Capitolo 5

Conclusione e considerazioni finali

In queste analisi è stato affrontato il problema dell'archiviazione efficiente dei dati genetici, prodotti tramite i sequenziatori NGS. Si è inoltre introdotta l'importanza dei K-mer e dei conteggi per la bioinformatica. Infine ci si è concentrati sui dati e sul salvataggio di questi, osservando due tool specializzati alla compressione dei dati genetici. Il salvataggio di queste enormi moli di dati risulta urgente, e quindi indispensabile da gestire, perché influisce negativamente sui costi di sequenziamento, mantenimento e distribuzione di questi dati. All'interno di queste analisi abbiamo comparato, in maniera efficace, l'output di due tool promettenti, sviluppati negli ultimi anni. Siamo riusciti ad ottenere valori validi che ci danno la possibilità di confrontarli efficacemente. Abbiamo quindi comparato i due tool, con occhio attento anche ai tool di compressione non specializzati, come riportato in tabella 4.3. Osservando i dati ottenuti(4.7) siamo, quindi, riusciti a concludere che il tool UST porta maggiori benefici in termini di compressione. Possiamo quindi ritenerci soddisfatti del risultato, sempre però, con una giusta interpretazione. Non possiamo infatti affermare che UST sia universalmente più pratico nell'utilizzo perché, in queste analisi, è stato valutato solo parzialmente il suo risultato. Non sono stati presi in considerazione, infatti, molti aspetti di questi tool come la facilità di utilizzo,l'affidabilità e neppure l'attendibilità dei dati ottenuti. Abbiamo quindi dato uno sguardo parziale ai tool, ma possiamo comunque affermare che UST, si prospetta più promettente.

Bibliografia

[1] Karel Břinda, Michael Baym e Gregory Kucherov. «Simplitigs as an efficient and scalable representation of de Bruijn graphs». In: *bioRxiv* (2020). DOI: 10.1101/2020.01.12.903443. eprint: https://www.biorxiv.org/content/early/2020/02/04/2020.01.12.903443.full.pdf. URL: https://www.biorxiv.org/content/early/2020/02/04/2020.01.12.903443.

[2] Nieves Brisaboa, Susana Ladra e Gonzalo Navarro. «DACs: Bringing direct access to variable-length codes». In: *Information Processing & Management* 49 (gen. 2013), pp. 392–404. DOI: 10.1016/j.ipm.2012.08.003.

[3] Rayan Chikhi, Antoine Limasset e Paul Medvedev. «Compacting de Bruijn graphs from sequencing data quickly and in low memory». In: *Bioinformatics* 32.12 (giu. 2016), pp. i201–i208. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btw279. eprint: https://academic.oup.com/bioinformatics/article-pdf/32/12/i201/17130531/btw279.pdf. URL: https://doi.org/10.1093/bioinformatics/btw279.

[4] Community. «GNU Gzip». eng. In: (2009). [Online; accessed 01-sept-2022]. URL: https://www.gnu.org/software/gzip/.

[5] Mikel Hernaez et al. «Genomic Data Compression». In: *Annual Review of Biomedical Data Science* 2.1 (2019), pp. 19–37. URL: https://doi.org/10.1146/annurev-biodatasci-072018-021229.

[6] Mikhail Karasikov et al. «Lossless Indexing with Counting de Bruijn Graphs». In: *bioRxiv* (2022). DOI: 10.1101/2021.11.09.467907. eprint: https://www.biorxiv.org/content/early/2022/02/03/2021.11.09.467907.full.pdf. URL: https://www.biorxiv.org/content/early/2022/02/03/2021.11.09.467907.

[7] Mikhail Karasikov et al. «MetaGraph: Indexing and Analysing Nucleotide Archives at Petabase-scale». In: *bioRxiv* (2020). DOI: 10.1101/2020.10.01.322164. eprint: https://www.biorxiv.org/content/early/2020/11/03/2020.10.01.322164.full.pdf. URL: https://www.biorxiv.org/content/early/2020/11/03/2020.10.01.322164.

[8] Ben Langmead. «Johns Hopkins University Algorithms for DNA Sequencing». eng. In: *Coursera* (2015). [Online; accessed 25-August-2022],https://youtu.be/hpb-mH-yjLc. URL: https://www.coursera.org/learn/dna-sequencing.

[9] Zhenyu Li et al. «Comparison of the two major classes of assembly algorithms: overlap–layout–consensus and de-bruijn-graph». In: *Briefings in Functional Genomics* 11.1 (dic. 2011), pp. 25–37. ISSN: 2041-2649. DOI: 10.1093/bfgp/elr035. eprint: https://academic.oup.com/bfg/article-pdf/11/1/25/473950/elr035.pdf. URL: https://doi.org/10.1093/bfgp/elr035.

[10] Armando Pinho e Diogo Pratas. «MFCompress: a compression tool for FASTA and multi-FASTA data». In: *Bioinformatics (Oxford, England)* 30 (ott. 2013). DOI: 10.1093/bioinformatics/btt594.

[11] Amatur Rahman e Paul Medvedev. «Representation of k-mer sets using spectrum-preserving string sets». In: *bioRxiv* (2020). DOI: 10.1101/2020.01.07.896928. eprint: https://www.biorxiv.org/content/early/2020/01/15/2020.01.07.896928.full.pdf. URL: https://www.biorxiv.org/content/early/2020/01/15/2020.01.07.896928.

[12] U.S. Government Services e Information. «National Library of Medicine». In: (). URL: https://www.ncbi.nlm.nih.gov/.

[13] Wikipedia contributors. *DNA — Wikipedia, The Free Encyclopedia*. [Online; accessed 29-August-2022]. URL: https://it.wikipedia.org/wiki/DNA.

[14] Wikipedia contributors. *K-mer — Wikipedia, The Free Encyclopedia*. [Online; accessed 1-September-2022]. URL: https://en.wikipedia.org/wiki/K-mer.

[15] Wikipedia contributors. *Next Generation Sequencing — Wikipedia, The Free Encyclopedia*. [Online; accessed 29-August-2022]. URL: https://it.wikipedia.org/wiki/Next_Generation_Sequencing.

[16] Wikipedia contributors. *Repeated sequence — Wikipedia, The Free Encyclopedia*. [Online; accessed 2-September-2022]. URL: https://en.wikipedia.org/wiki/Repeated_sequence_(DNA).