Università degli Studi di Padova
Dipartimento di ingegneria dell'Informazione

Corso di Laurea in Ingegneria Informatica

Analisi di sequenze di SARS-CoV-2

Costruzione e confronto di alberi filogenetici

Gruppo di lavoro:
Tommaso Green, tommaso.green@studenti.unipd.it
Matricola 1236296

Nicola Gulmini, nicola.gulmini@studenti.unipd.it
Matricola 1237216

Giulia Pezzutti, giulia.pezzutti@studenti.unipd.it
Matricola 1234012

Docente:
Prof. Matteo Comin

Anno Accademico 2019-2020

Sommario

Con la recente pandemia di COVID-19 causata dal virus SARS-CoV-2, è aumentato l'interesse nei confronti di tale virus, e più nello specifico nelle dinamiche della sua evoluzione e diffusione.

La disponibilità di genomi virali sequenziati, provenienti da diverse parti del mondo e catalogati in banche dati pubbliche e ben fornite, permette alla bioinformatica di intervenire a supporto di analisi filogenetiche e non, introducendo, però, anche ulteriori sfide.

In questo lavoro sono confrontate alcune sequenze di SARS-CoV-2 opportunamente selezionate, e prodotti degli alberi filogenetici a partire dai diversi metodi di confronto adottati. Infine, una discussione dei risultati ottenuti permette di apprezzare le differenze tra i diversi percorsi intrapresi, evidenziandone i pro e i contro.

Con l'intento di produrre un lavoro comprensibile e facilmente estendibile, lungo l'intero elaborato è stato mantenuto un costante grado di approfondimento per ciascun aspetto della computazione eseguita: dalla teoria alla base degli algoritmi, alle specifiche interessanti dei tool che li implementano, fino alla discussione dei risultati.

Indice

Capitolo 1

Introduzione

L'interesse nei confronti del virus SARS-CoV-2 deriva dall'attuale emergenza sanitaria dovuta alla *pandemia di COVID-19 del 2019-2020*, e dalla grande disponibilità di dati relativi a essa provenienti da diverse parti del mondo. Sorge quindi l'interesse per l'analisi filogenetica, per portare ordine tra tali dati, la cui correlazione è importante quanto difficile da individuare.

1.1 SARS-CoV-2

Il SARS-CoV-2 (*Severe Acute Respiratory Syndrome-Related Coronavirus 2*) è un ceppo virale scoperto alla fine del 2019, il cui genoma è costituito da un filamento di RNA di taglia circa 27-32 *kb* (migliaia di basi) [42]. Secondo la classificazione di Baltimore esso fa parte del gruppo IV, secondo la classificazione ICTV esso appartiene alla specie SARS-related coronavirus, genere Betacoronavirus, famiglia Coronaviridae, sottogenere Sarbecovirus.

Esso è stato identificato come la causa di un'epidemia di malattia respiratoria rilevata per le prima volta a Wuhan, in Cina, a dicembre del 2019. La malattia causata da questo virus è denominata *coronavirus disease 2019* (COVID-19). Nonostante le misure restrittive adottate, il virus ha potuto diffondersi in tutto il mondo, portando al lockdown in centinaia di paesi.

1.1.1 Mutazioni

Una cellula infettata dal virus rilascia milioni di nuovi virus, portatori di copie del genoma originale. Nella duplicazione del genoma possono essere commessi alcuni errori, le mutazioni, e man mano che il virus si diffonde, le mutazioni si accumulano nei nuovi individui. Le mutazioni sono causate da diversi fattori e sono di diverso tipo; possono inoltre alterarne le proprietà oppure lasciarle invariate [46]. Le mutazioni del genoma spesso cambiano i caratteri contenuti in un gene senza però modificarne la proteina codificata, poiché in generale lo stesso aminoacido può essere codificato da più geni:

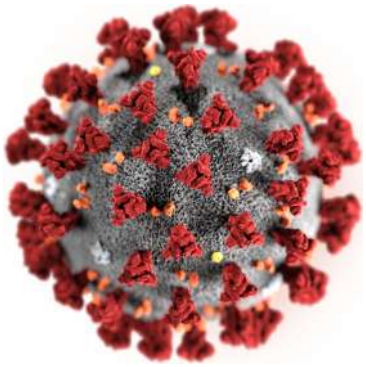


Figura 1.1: Illustrazione realizzata dai CDC [1] che mostra la morfologia del virus. Gli elementi di colore rosso, arancione e giallo, sono proteine.

queste mutazioni sono dette silenziose. Le mutazioni non silenziose, invece, provocano anche un cambiamento nella sequenza proteica codificata: in virtù del fatto che le proteine possono essere costituite anche da migliaia di aminoacidi, non sempre questa mutazione ha un effetto evidente.

A questo punto della pandemia, i genomi di SARS-CoV-2 con 10 o meno mutazioni sono comuni, e solo un piccolo numero ne ha più di 20, ancora corrispondente a meno di un decimo del genoma.

Alcuni studi hanno dimostrato come il virus muti con un ritmo regolare e come ciò permetta quindi di condurre delle analisi filogenetiche per ricostruirne l'evoluzione: l'accumulo di mutazioni, infatti, permette di tracciare un percorso spaziale e temporale della diffusione del virus.

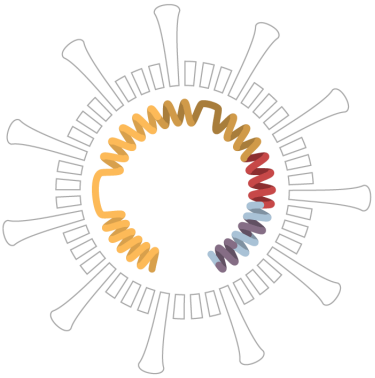


Figura 1.2: Rappresentazione del virus, con la struttura a corona e il filamento di RNA all'interno [46].

Capitolo 2

Confrontare sequenze

Il confronto di sequenze genomiche o proteiche è uno dei maggiori obiettivi della bioinformatica, poiché permette di determinare similarità e dissimilarità tra le sequenze stesse, e di conseguenza inferirne considerazioni di tipo funzionale.

Se le sequenze analizzate sono di individui della stessa specie, è possibile individuare la correlazione tra alcune caratteristiche degli individui, mutazioni o tracciarne l'evoluzione. Per quest'ultima, uno strumento molto utile risulta l'albero filogenetico, un diagramma che permette di rappresentare e visualizzare il processo evolutivo d'interesse.

Questo capitolo si apre con la notazione che sarà seguita coerentemente in tutto l'elaborato. Sono poi descritti alcuni metodi di confronto tra sequenze, costruzione e confronto tra alberi filogenetici, con una sezione dedicata al clustering effettuato a partire dagli alberi stessi.

2.1 Notazione

Anche se il DNA è una molecola flessibile e tridimensionale che interagisce in un ambiente dinamico, la sua informazione digitale può essere rappresentata da una stringa di caratteri A, C, G e T [32].

Sarà indicato con Σ un alfabeto finito ed arbitrario e con $\Omega = \{A, C, G, T\}$ l'alfabeto utilizzato all'interno delle sequenze genomiche. Molti degli strumenti matematici presentati nel seguito si applicano a qualsiasi tipo di sequenze, ma è nostro interesse applicarli a sequenze genomiche, per cui ci limiteremo quasi sempre a esse.

Tipicamente, le sequenze genomiche sono indicate con delle lettere maiuscole, ad esempio A, B, X, Y, e con $\|\cdot\|$ si indica la loro lunghezza; la scrittura $X \in \Omega^{n>0}$ significa considerare una successione di simboli $X_{i\in[1,n]} \in \Omega$, $X = X_1X_2 \dots X_n$ di lunghezza $\|X\| = n \in \mathbb{N}$. Per indicare la lunghezza delle sequenze sono utilizzate delle lettere minuscole, quasi sempre n, m. La lettera k è preferita per indicare la lunghezza dei k-mer, piccole porzioni

di sequenze, mentre con w si indicano le parole o sottostringhe $w_1 \dots w_L$ di lunghezza generalmente $k \leq L \ll n$.

Altre definizioni e la relativa notazione saranno fornite nel corso della lettura, per essere utilizzate quando se ne presenterà la necessità.

2.2 Allineamento

L'allineamento è il metodo concettualmente più immediato e di facile comprensione utilizzabile per confrontare sequenze genomiche. Date due sequenze definite sull'alfabeto Ω, esso mira a incolonnare i caratteri che le compongono, mantenendone l'ordine, in modo da ottenere il maggior numero possibile di corrispondenze. La corrispondenza tra i caratteri allineati è definita *match*, mentre se due caratteri vengono incolonnati pur essendo diversi, si è in presenza di un *mismatch*. A volte si rivela necessario inserire, tra i simboli di una sequenza, un carattere vuoto. Tale operazione, chiamata *indel*, rappresenta l'assenza del carattere corrispondente nell'altra sequenza, ed è generalmente indicata con un tratto orizzontale: "$-$" oppure "_". Associando uno specifico *score* a ciascuna delle operazioni appena definite, è possibile valutare un intero allineamento e determinare quello ottimo. Gli score sono tipicamente raccolti, nel caso più semplice di un allineamento di una sola coppia di sequenze, in una matrice di score δ di dimensione $(\|\Omega\| + 1) \times (\|\Omega\| + 1)$. Ogni elemento della matrice (i, j) contiene lo score da assegnare ai caratteri i e j quando essi vengono allineati. Tale matrice risulta simmetrica rispetto alla diagonale principale: nella diagonale sono presenti gli score da attribuire ai match, nelle altre posizioni gli score per i mismatch, mentre l'ultima riga e l'ultima colonna della matrice sono dedicate agli score degli indel (l'ultimo elemento in basso a destra è inutilizzato, poiché non ha senso allineare indel). A seconda di alcune osservazioni o informazioni aggiuntive circa l'analisi condotta, i valori di score possono variare.

L'allineamento, quando è tra due sequenze, è chiamato *pairwise*. La sua estensione, l'allineamento *multiplo*, prevede di considerare contemporaneamente più sequenze.

Quando l'allineamento considera le sequenze nella loro interezza è detto *globale*, quando ne considera solo delle porzioni è detto *locale*. Nel seguito ci concentreremo sempre su allineamenti globali.

2.2.1 Algoritmo di Needleman e Wunsch

L'algoritmo di Needleman e Wunsch è un algoritmo di programmazione dinamica con il quale, data in ingresso una coppia di sequenze, se ne possono determinare in modo esatto l'allineamento ottimo e lo score associato.

Siano $X \in \Omega^{n>0}$ e $Y \in \Omega^{m>0}$ le due sequenze in input. Si costruisce una griglia $(n+1) \times (m+1)$: la prima riga e la prima colonna, considerate di indice 0, rappresentano l'inizio delle due sequenze mentre ciascuna riga corrisponde a ciascun carattere della prima sequenza, e ciascuna colonna corrisponde a un carattere della seconda sequenza. Nelle celle con $i = 0$ o $j = 0$ viene sempre memorizzato valore 0. In ogni cella (i, j), con $i \in [1, n]$ e $j \in [1, m]$, si memorizza lo score $s_{i,j}$ dell'allineamento ottimo tra le sottosequenze $X_1 \dots X_i$ e $Y_1 \dots Y_j$. Il calcolo è possibile noti gli score degli allineamenti ottimi $s_{i-1,j-1}$, $s_{i-1,j}$ e $s_{i,j-1}$ e la matrice di score δ. Si realizzando quindi tre possibili casi:

1. I caratteri X_i e Y_j si allineano con un match o con un mismatch: aggiungendo un carattere all'allineamento in entrambe le sequenze, lo score è determinato dalla somma dello score $s_{i-1,j-1}$ dell'allineamento tra $X_1 \dots X_{i-1}$ e $Y_1 \dots Y_{j-1}$ e lo score $\delta(X_i, Y_j)$ dell'allineamento dei singoli caratteri, presente nella matrice di score;

2. Viene aggiunto un carattere solo nella sequenza X, cioè il carattere X_i è allineato con il carattere "_" nella sequenza Y. Questo tipo di indel è detto cancellazione. In questo caso lo score è dato dalla somma dei contributi dell'allineamento tra le sequenze $X_1 \dots X_{i-1}$ e $Y_1 \dots Y_j$, chiamato $s_{i-1,j}$, e dell'allineamento del carattere X_i con un "_", ovvero nella matrice di score $\delta(X_i, \_)$;

3. Viene aggiunto un carattere solo nella sequenza Y, ovvero il carattere Y_j viene allineato con "_". Questo tipo di indel è detto inserimento. Si procede in modo analogo al caso precedente, sommando $s_{i,j-1}$ con $\delta(\_, Y_j)$.

Dei tre valori di score calcolati se ne seleziona il massimo poichè si cerca sempre l'allineamento di score massimo. Di fatto, lo score inserito nella cella (i, j) è determinato secondo la seguente relazione di ricorrenza.

$$s_{i,j} = \max \begin{cases} s_{i-1,j-1} + \delta(X_i, Y_j), \\ s_{i-1,j} + \delta(X_i, \_), \\ s_{i,j-1} + \delta(\_, Y_j) \end{cases} \tag{2.1}$$

L'ultimo elemento calcolato nella griglia è l'elemento (n, m), il quale per definizione contiene lo score dell'allineamento ottimo per le due intere sequenze.

É possibile memorizzare a ogni passo dell'algoritmo anche come si è calcolato ciascun valore, memorizzando, in corrispondenza di ogni elemento della matrice, la direzione di provenienza del nodo che ha generato il valore ottimo. Così facendo, una volta calcolati gli score di tutte le celle, si può seguire il percorso partendo dal nodo (n, m) e determinato dalle direzioni memorizzate, per ricostruire l'effettivo allineamento ottimo.

Si riporta l'algoritmo per la costruzione della matrice di allineamento con la memorizzazione delle direzioni.

Algoritmo di Needleman-Wunsch

for $i \leftarrow 0$ **to** n **do**
  $s_{i,0} \leftarrow 0$
for $j \leftarrow 1$ **to** m **do**
  $s_{0,j} \leftarrow 0$
for $i \leftarrow 1$ **to** n **do**
  for $j \leftarrow 1$ **to** m **do**

$$s_{i,j} = \max \begin{cases} s_{i-1,j-1} + \delta(X_i, Y_j) \\ s_{i-1,j} + \delta(X_i, \_) \\ s_{i,j-1} + \delta(\_, Y_j) \end{cases}$$

$$b_{i,j} = \begin{cases} \text{“}\uparrow\text{”} & s_{i,j} = s_{i-1,j} + \delta(X_i, \_) \\ \text{“}\leftarrow\text{”} & s_{i,j} = s_{i,j-1} + \delta(\_, Y_j) \\ \text{“}\nwarrow\text{”} & s_{i,j} = s_{i-1,j-1} + \delta(X_i, Y_j) \end{cases}$$

return $s_{n,m}, \mathbf{b}$

Esempio 1. In figura 2.1 e 2.2 sono presenti la griglia e il risultato dell'algoritmo su due semplici sequenze, usando l'identità come matrice di score: ai match è assegnato score +1 e agli indel e mismatch è associato valore 0. Con una prima scansione vengono inseriti i valori all'interno della griglia seguendo la relazione di ricorrenza della formula 2.1 e memorizzando la direzione dalla quale è stato ricavato il valore. Con una seconda scansione, partendo dall'elemento in basso a destra e seguendo le direzioni delle frecce, si ricostruisce l'allineamento ottimo.

L'algoritmo di Needleman e Wunsch ha complessità temporale e spaziale $O(\|X\| \times \|Y\|) = O(nm)$: la matrice ha per costruzione dimensione $(n + 1) \times (m + 1)$ e per memorizzarne i valori è necessario scandirla interamente una volta tramite operazioni tempo costante. La seconda scansione per la ricostruzione dell'allineamento richiede un numero $O(n + m) \in O(nm)$ di operazioni tempo costante.

Tale algoritmo è stato successivamente esteso con l'introduzione dei *gap opening penalties* e *gap extension penalties*: in natura si è osservato come fosse più frequente avere un indel composto da più caratteri piuttosto di un numero maggiore di indel di singoli caratteri. In tal senso, il concetto di indel può essere interpretato come una prima azione di apertura della sequenza per l'inserimento e una successiva azione per inserimento di caratteri all'interno di essa. Il costo per la presenza di ciascun indel formato da x inserimenti successivi viene così riportato alla somma di un fattore, chiamato gap opening penalty $-\rho$, per l'apertura della stringa e un fattore dato dal prodotto della lunghezza dell'indel per il costo, chiamato gap extension penalty $-\sigma$, di ciascun inserimento. In conclusione, il costo di un indel di

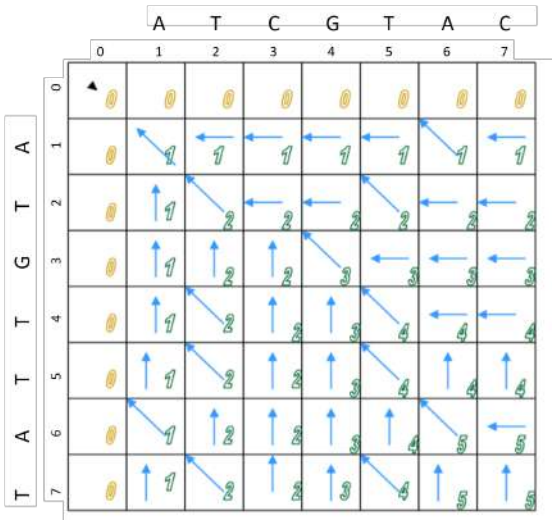


Figura 2.1: Griglia ottenuta dopo la prima scansione dell'algoritmo sulle sequenze $X = ATGTTAT$ e $Y = ATCGTAC$. Le frecce azzurre indicano la direzione che permette di ottenere l'ottimo, i numeri in verde sono gli score associati a ciascun allineamento. Se i o j sono pari a 0, non si sta facendo allineamento in quanto si considera solo una sequenza, dunque lo score è nullo.

AT-GTTA-T
ATCG-TAC-

Figura 2.2: Allineamento ottimo ricostruito seguendo le frecce nella griglia, che mostra i caratteri delle due sequenze correttamente incolonnati.

x caratteri è calcolato come $-(\rho + x \cdot \sigma)$, con $\rho, \sigma > 0$ e generalmente, per le considerazioni sulla naturale dinamica di tale fenomeno, $\rho \gg \sigma$.

Tale estensione può essere implementata anche nella matrice degli score: la singola matrice viene convertita in tre allineate una sopra l'altra; la matrice centrale è utilizzata per gli spostamenti dovuti a match o mismatch e per essa si applicano gli score dovuti alla weight matrix, mentre le altre due matrici indicano gli score dovuti ad inserimenti in una o nell'altra sequenza. Partendo dalla matrice centrale, per posizionare si deve considerare il costo per spostarsi da tale matrice ad un altra, pari al gap opening penalty e è successivamente possibile muoversi in tali matrici pagando, per ciascun passo, un costo pari al valore di gap extension penalty. Per tornare alla matrice centrale, nel caso si abbiano match o indels, non è invece necessario pagare alcun costo. Con tale nuova procedura, si processa nuovamente la matrice per righe o per colonne con lo scopo di determinare nuovamente l'allineamento ottimo.

2.2.2 Algoritmo di Wilbur e Lipman

L'algoritmo di Wilbur e Lipman è un algoritmo euristico che migliora la complessità temporale rispetto all'algoritmo di Needleman e Wunsch: esso

si basa infatti sull'idea che per avere un buon allineamento globale è necessario osservare buoni allineamenti locali di lunghezza inferiore [50]. Tale algoritmo infatti si limita a considerare singoli k-mer anziché studiare le intere sequenze. Il parametro k, lunghezza dei k-mer considerati, è uno dei parametri forniti in ingresso all'algoritmo e all'aumentare di tale parametro, l'algoritmo diviene più veloce ma più approssimato; tipicamente, per sequenze di DNA, esso è pari a 4. Sicuramente tale valore deve essere minore della minima lunghezza delle due sequenze in ingresso.

L'algoritmo riceve in ingresso le due sequenze, $X \in \Omega^{n>0}$ e $Y \in \Omega^{m>0}$ e inizialmente cerca di determinare i match tra i k-mer $\in \Omega^k$ delle due sequenze, ovvero cerca tutti i k-mer $X_iX_{i+1}\dots X_{i+k-1}$ nella prima sequenza e tutti i k-mer $Y_jY_{j+1}\dots Y_{j+k-1}$ nella seconda sequenza tali che

$$X_iX_{i+1}\dots X_{i+k-1} = Y_jY_{j+1}\dots Y_{j+k-1}.$$

I match possono essere determinati in modo efficiente, ovvero in tempo $O(\max\{n, m\})$, tramite all'utilizzo di due array di lunghezza pari a $\|\Omega\|^k$, il numero di possibili differenti k-mer. Considerando ad esempio la prima sequenza, ogni sua posizione i caratterizza l'inizio di un k-mer, il quale può essere codificato con un codice numerico ic (tramite una codifica definita a priori): si definisce quindi un vettore C in cui in ogni posizione ic viene memorizzata la lista delle posizioni della sequenza in cui è possibile trovare i k-mer con codifica ic.

Il non determinismo dell'algoritmo compare nell'utilizzo di tali k-mer per la costruzione dell'allineamento: non si considerano tutti i match di k-mer ma solo quelli considerati più significativi secondo quanto ora descritto.

Considerando un match di k-mer, esso inizia in posizione i nella prima sequenza e in posizione j nella seconda sequenza. Si può calcolare la *diagonale* d_p con $p \in [1-m, n-1]$, definita anche *offset*, in cui si trova tale coppia di k-mer: si dice che il match di k-mer appartiene alla diagonale d_p se e solo $p = i - j$. Per costruzione, si può affermare che sicuramente ogni match di k-mer appartiene a una certa diagonale. Si definisce inoltre distanza tra le due diagonali d_p e d_q il valore $|q - p|$.

Per la computazione di tali diagonali, è conveniente memorizzare un vettore d, inizialmente nullo, in cui ogni qualvolta si determina che un match di k-mer appartenente ad una certa diagonale d_p, si incrementa di uno il valore contenuto in $d[p]$.

Determinate le diagonali è possibile determinare la media e la deviazione standard del numero di match contenuti nelle diagonali, ovvero dei valori contenuti nell'arra. Grazie a tali valori, si definiscono *significative* le diagonali che contengono un numero di k-mer maggiore del valore della media sommato alla deviazione standard. Si definisce a questo punto il *window space* l'insieme di diagonali a distanza al massimo $w \in \mathbb{N}$ da ciascuna diagonale significativa, dove w è un valore fornito in ingresso all'algoritmo. Per

la computazione si considerano quindi solo le diagonali che contengono un elevato numero di match, per analizzare solo delle piccole regioni simili.

Partendo dai k-mer all'interno del window space, si può ora applicare l'algoritmo di Needleman-Wunsch, descritto in 2.2.1, per determinare un allineamento ottimo: in particolare, si applica il metodo di scoring che associa un valore $+1$ a ciascun match di caratteri all'interno dei k-mer precedentemente determinati mentre i match di caratteri non compresi nei k-mer non vengono considerati e infine si associa un valore $-g$ a gap di qualsiasi lunghezza. Grazie a tale costruzione, la matrice viene determinata direttamente intorno ai match di k-mer nella zona corrispondente al window space e non è più necessaria l'analisi dell'intera matrice $(n+1) \times (m+1)$, come nel caso standard dell'algoritmo di Needleman-Wunsch.

L'algoritmo può avere $k = 1$ e comprendere nel window space tutte le possibili diagonali, quindi tutti i possibili allineamenti tra le due sequenze: in questo caso, la complessità e i risultati coincidono con l'algoritmo di Needleman e Wunsch; considerando invece valori minori di w e valori maggiori di uno per k, si ottiene un risultato più veloce ma approssimato. Per tali motivazioni, complessità e accuratezza dell'algoritmo dipendono dalla lunghezza della stringa e dai parametri k e w.

Tale metodo verrà successivamente implementato, con piccole modifiche che ne permettano l'adattabilità a una serie di diversi problemi nella ricerca del miglior allineamento, nell'algoritmo FastA.

2.2.3 Soluzioni efficienti

In generale, il problema dell'allineamento globale risulta particolarmente difficile non in termini di complessità temporale bensì di spazio richiesto. Nel caso del problema in esame sembrerebbe necessario dover memorizzare tutti gli $O(nm)$ score degli allineamenti dei prefissi delle stringhe in input, come visto nella sezione 2.2.1.L'idea è quindi di sviluppare un approccio che richieda spazio lineare nella taglia della sequenza più lunga cioè $O(n)$. Prima di procedere è necessario introdurre una struttura matematica ausiliaria per l'allineamento.

Date le sequenze di input $X = X_1X_2 \dots X_n$ e $Y = Y_1Y_2 \dots Y_m$ l'*edit graph* è un grafo $G = (V, E)$ dove i vertici sono associati a coppie di interi (i, j) con $i \in [0, n]$ e $j \in [0, m]$. Esso è solitamente rappresentato in una struttura a griglia costituita da $n + 1$ righe (una per ogni carattere X_i e una riga 0 ausiliaria) e da $m + 1$ colonne (una per ogni carattere Y_j e una colonna 0 ausiliaria). In un arbitrario vertice (i, j) vi sono tre archi entranti ciascuno corrispondente ad una colonna della matrice di allineamento:

- un arco diagonale associato alla colonna $\binom{X_i}{Y_j}$;

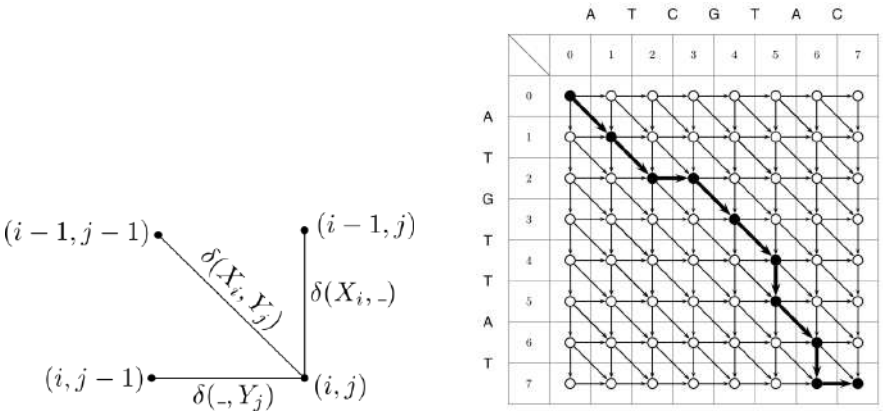


Figura 2.3: Archi entranti in un nodo (i, j) (a sinistra) ed esempio di edit graph (a destra)

- un arco orizzontale associato alla colonna $\begin{pmatrix} - \\ Y_j \end{pmatrix}$;
- un arco verticale associato alla colonna $\begin{pmatrix} X_i \\ - \end{pmatrix}$.

Grazie a questa associazione, si può pensare che ogni arco sia legato al relativo score dato dalla scoring matrix δ. Ad ogni allineamento corrisponde quindi un *path* nell'edit graph che parte dal vertice $(0, 0)$ detto *source* e arriva al vertice (n, m) detto *sink*. Tale path avrà uno score associato $s_{n,m}$ dato dalla somma degli score degli archi che prendono parte al path stesso. Di conseguenza, l'allineamento ottimo corrisponde al path che massimizza lo score, cioè il path più lungo.

Il concetto di edit graph è essenziale per introdurre il concetto di *affine gap penalties*. Se si associa un valore costante σ in presenza di un indel, in presenza di un *gap* di x indel consecutivi e adiacenti, lo score è penalizzato di una quantità pari a σx. Tale penalizzazione risulta eccessiva data la frequenza con la quale avvengono le mutazioni in una sequenza genomica, per questo si decide di penalizzare di più l'apertura di un gap, piuttosto che la presenza di indel adiacenti. Si adotta, perciò, una funzione affine $\rho + \sigma x$ dove $\rho > 0$ rappresenta la penalità di apertura del gap, mentre $\sigma > 0$ tiene conto dell'estensione del gap stesso, con $\rho \gg \sigma$. Le affine gap penalties sono rappresentate nell'edit graph aggiungendo per ogni nodo (i, j) un arco per ogni nodo a sud $(i+x, j)$ con uno score associato di $-(\rho+\sigma x)$ e analogamente un arco per ogni nodo ad est $(i, j + x)$ avente score $-(\rho + \sigma x)$.

Poiché la ricerca del path di score massimo in un grafo è dell'ordine del numero di archi, ed essendo questi aumentati di un fattore $O(n)$, la complessità totale diventa $O(n^3)$. Per evitare che ciò accada, è sufficiente

suddividere la relazione di ricorrenza degli score in tre forme distinte:

$$\overset{\downarrow}{s}_{i,j} = \max \begin{cases} \overset{\downarrow}{s}_{i-1,j} - \sigma, \\ s_{i-1,j} - (\rho + \sigma) \end{cases}$$

$$\vec{s}_{i,j} = \max \begin{cases} \vec{s}_{i,j-1} - \sigma, \\ s_{i,j-1} - (\rho + \sigma) \end{cases}$$

$$s_{i,j} = \max \begin{cases} s_{i-1,j-1} + \delta(X_i, Y_j), \\ \overset{\downarrow}{s}_{i,j}, \\ \vec{s}_{i,j} \end{cases}$$

La variabile $\overset{\downarrow}{s}_{i,j}$ tiene conto dello score dell'allineamento del prefisso i-esimo di X ($X = X_1X_2 \dots X_i$) e il prefisso j-esimo di Y ($Y = Y_1Y_2 \dots Y_j$) che termina con un gap in Y, mentre la variabile $\vec{s}_{i,j}$ rappresenta l'analogo allineamento che termina con un gap in X. Infatti per entrambe le variabili il primo termine della ricorrenza rappresenta l'apertura del gap mentre il secondo l'estensione dello stesso.

Si presentano ora soluzioni efficienti in termini di complessità temporale e spaziale per l'allineamento.

Calcolo dello score con spazio lineare

Si può osservare facilmente dalle relazioni di ricorrenza ricavate che un valore arbitrario di score $s_{i,j}$ dipende solamente dagli score:

- $s_{i-1,j}$, cioè lo score nella stessa colonna e nella riga precedente;
- $s_{i,j-1}$, cioè lo score nella stessa riga e nella colonna precedente;
- $s_{i-1,j-1}$, cioè lo score immediatamente precedente nella diagonale.

Di conseguenza, volendo produrre in output esclusivamente lo score dell'allineamento ottimo, è sufficiente mantenere, durante la computazione, due vettori di lunghezza n che conservano le informazioni riguardanti la colonna analizzata e quella immediatamente precedente. Questo accorgimento, tuttavia, non consente di ottenere l'allineamento ottimo in spazio lineare dato che, com'è tipico della programmazione dinamica, è necessario memorizzare per ogni nodo (i, j) dell'edit graph un valore di backtracking che consenta di ricostruire, a partire dal nodo (n, m), l'allineamento ottimo. Siccome tutti i valori di backtracking devono essere salvati, lo spazio rimane $O(nm)$.

Divide and conquer

La soluzione proposta da Hirschberg [35] nel 1975 per il problema della *LCS*, consiste nell'abbandonare il paradigma della programmazione dinamica a favore di un approccio divide and conquer.

Come si è detto, l'allineamento ottimo corrisponde a un path di score massimo nell'edit graph che parte dal vertice source $(0, 0)$ e termina nel vertice sink (n, m). Esso è vincolato a passare anche per un incognito vertice intermedio detto *middle* $(mid, \frac{m}{2})$ (sia m pari per semplicità) situato in qualche punto sulla colonna di indice $\frac{m}{2}$. Per trovare il valore ignoto mid si definisce, per ogni $i \in [0, n]$, $length(i)$ la lunghezza del path di score più elevato da source a sink che passa per il punto $(i, \frac{m}{2})$. Poiché della colonna $\frac{m}{2}$-esima il punto $(mid, \frac{m}{2})$ è quello che massimizza la lunghezza del path da source a sink, si ha facilmente che

$$length(mid) = \max_{i \in [0,n]} length(i).$$

Si deve ora sviluppare un procedimento per il calcolo efficiente di $length(i)$ senza conoscere il path di score massimo e usando memoria lineare. S'osservi innanzitutto che un arbitrario vertice $(i, \frac{m}{2})$ divide il path di lunghezza $length(i)$ in due parti:

1. un path dal source a $(i, \frac{m}{2})$, la cui lunghezza è chiamata $prefix(i) = s_{i,\frac{m}{2}}$;
2. un path da $(i, \frac{m}{2})$ al sink, di lunghezza $suffix(i) = s^{reverse}_{i,\frac{m}{2}}$, poiché rappresenta la lunghezza del path di score massimo dal $(n, m) \to (i, \frac{m}{2})$ nell'edit graph con gli archi in senso inverso.

Vale quindi

$$length(i) = prefix(i) + suffix(i) = s_{i,\frac{m}{2}} + s^{reverse}_{i,\frac{m}{2}}. \tag{2.2}$$

Grazie alla relazione 2.2 è possibile trovare il punto medio mid in spazio lineare, e in un tempo proporzionale all'area del rettangolo a sinistra della colonna $\frac{m}{2}$ e del rettangolo a destra di essa, cioè $O(nm)$. Una volta trovato il punto $(mid, \frac{m}{2})$, il problema originale si divide in due sottoproblemi di taglia inferiore: trovare i path di score massimo $(0, 0) \to (mid, \frac{m}{2})$ e $(mid, \frac{m}{2}) \to (n, m)$. Questa volta la complessità temporale è pari alla somma della metà dei due rettangoli precedentemente menzionati. Proseguendo ricorsivamente, la complessità temporale del problema sarà proporzionale alla somma: $area + \frac{area}{2} + \frac{area}{4} + \frac{area}{8} + \cdots \leq 2 \cdot area$, dove $area$ è la somma dei due rettangoli iniziali (cfr. figura **??**). Infine, raccolti tutti i punti intermedi, è possibile trovare l'allineamento ottimo. La complessità temporale aumenta di un fattore 2 tuttavia lo spazio totale richiesto diviene $O(n)$ passando da quadratico a lineare.

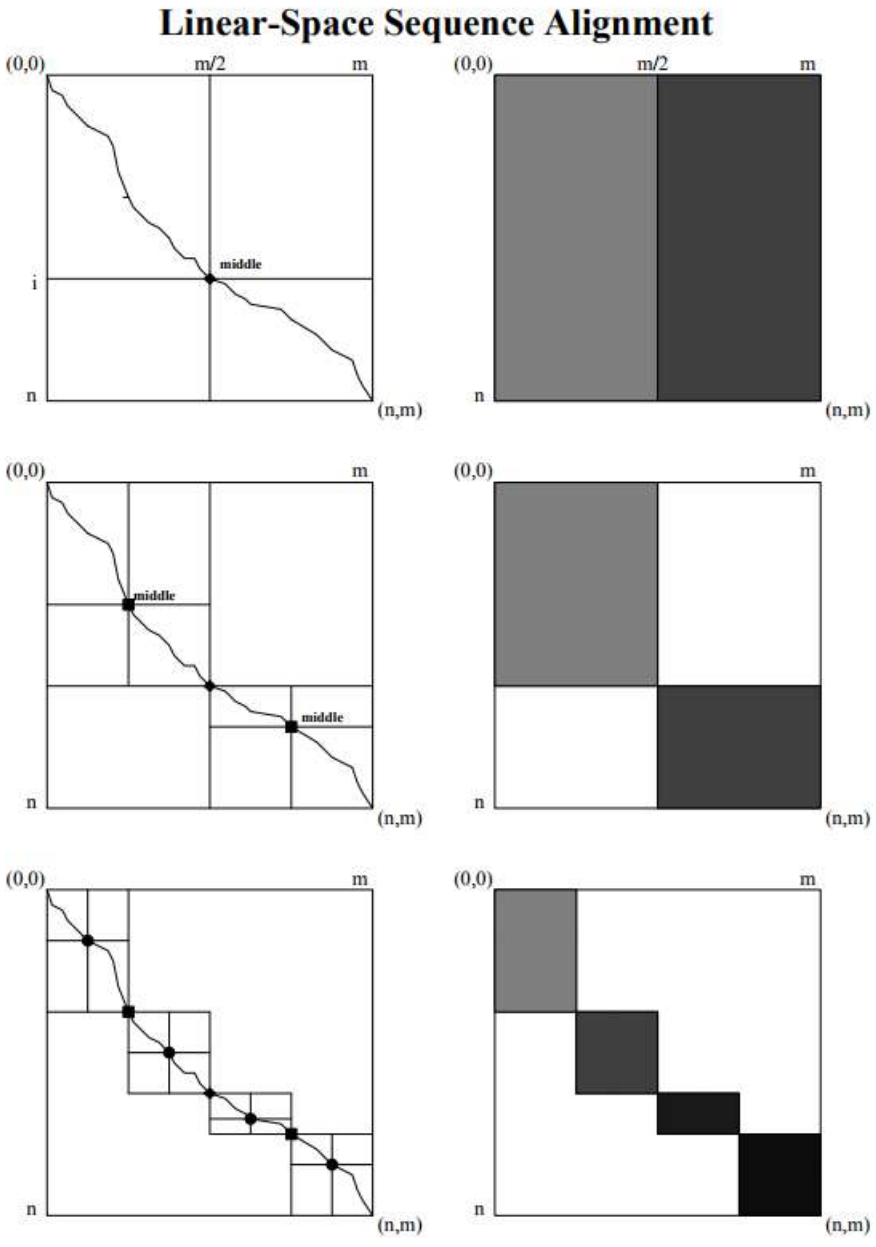


Figura 2.4: Stato dell'algoritmo divide and conquer in iterazioni successive: a sinistra i middle point trovati, a destra il tempo richiesto in funzione dell'area della tabella coperta

Ulteriori sviluppi

Myers e Miller [40], riprendendo questa idea originale di Hirschberg e adattandola all'algoritmo di allineamento globale di Gotoh [34] (algoritmo che funziona solo in caso di affine gap penalties), propongono un algoritmo efficiente di allineamento in spazio lineare, alla base del pacchetto `stretcher` di EMBOSS, trattato nella sezione 3.2.

2.2.4 Allineamento multiplo progressivo

L'allineamento multiplo è la naturale generalizzazione dell'allineamento pairwise, poiché prende in considerazione più di due sequenze da allineare. L'obiettivo rimane quello di massimizzare lo score, ma l'aumento della difficoltà computazionale del problema richiede di adottare un approccio diverso per la sua risoluzione.

Si osservi che un allineamento globale di $N > 2$ sequenze induce $\binom{N}{2}$ allineamenti pairwise e che partendo da altrettanti allineamenti pairwise è possibile costruire un allineamento globale. Questo principio è alla base dell'*allineamento progressivo*: sfruttando quanto già visto nelle sezioni 2.2.1 e 2.2.2 è possibile partire da una coppia di sequenze e aggiungere una sequenza alla volta, iterativamente, fino a ricostruire l'intero allineamenoto multiplo.

Va sottolineato il fatto che partendo da un allineamento multiplo ottimo, gli allineamenti pairwise indotti non sono in generale ottimi; viceversa, partire da allineamenti pairwise ottimi non garantisce di ottenere un allineamento multiplo ottimo per problemi di incompatibilità. Per questo motivo, l'ordine con il quale si selezionano le sequenze negli allineamenti pairwise incide considerevolmente sulla qualità dell'allineamento multiplo finale.

Approccio greedy

Dopo aver determinato lo score degli allineamenti di tutte le possibili coppie di sequenze, vengono selezionate le due sequenze con il maggiore livello di similarità. Tali sequenze vengono allineate tramite un algoritmo di allineamento pairwise e se ne determina un *vettore profilo*, lungo al più come l'allineamento tra le due sequenze. Nella generica posizione i di tale vettore, nel caso di un match tra le due sequenze in tale posizione, si memorizza il carattere che lo ha generato; nel caso di un mismatch o un indel nella posizione i, vengono riportati nel vettore profilo entrambi i possibili caratteri.

Il vettore profilo genera successivamente il vettore *consensus*: esso è in generale definito come la sequenza che contiene, in ogni posizione i, il carattere che appare in modo più frequente nella posizione i delle sequenze analizzate. Se, però, in una stessa posizione sono presenti più caratteri ugualmente frequenti, se ne sceglie uno arbitrariamente.

Il vettore consensus così creato riassume le informazioni delle due sequenze di partenza e del relativo allineamento e può, per tale ragione, sostituirle nell'insieme delle sequenze da analizzare.

A questo punto, è necessario calcolare le distanze tra la nuova sequenza consensus e le altre presenti nell'interno dell'insieme iniziale. L'algoritmo procede poi iterativamente ripetendo quanto appena descritto: si seleziona la coppia di sequenze più vicine, si allineano e si sostituiscono con la relativa sequenza consensus, diminuendo di 1 la cardinalità dell'insieme di sequenze da confrontare. L'algoritmo termina una volta allineate tutte le sequenze, cioè quando si determina la sequenza consensus che rappresenta l'intero insieme di sequenze in ingresso.

Star approach

É possibile utilizzare altre tecniche per la scelta della coppia di sequenze da analizzare, una di queste è detta *star approach*. Si sceglie una sequenza come centro dell'allineamento e si allineano tutte le altre a essa. Tali allineamenti pairwise sono successivamente allineati tra loro per formare l'allineamento multiplo. Questa tecnica non è affetta da problemi di compatibilità tra le sequenze, dato che tutti gli allineamenti hanno una sequenza comune, ma la scelta del centro risulta determinante per il risultato. Per ottenere lo score massimo si possono testare tutte le sequenze come centro e poi selezionare quella che fornisce il risultato migliore.

Tree approach

Un'altra tecnica prevede l'utilizzo di un *albero guida*: si compiono gli allineamenti pairwise tra tutte le coppie di sequenze, si produce una matrice che ne conserva gli score e da questa si costruisce un albero binario (si veda la sezione 2.4.1). Tale albero induce un ordine con il quale considerare le sequenze nell'allineamento progressivo, risolvendo il problema dello star approach.

L'allineamento progressivo in pratica funziona molto bene per allineamenti di sequenze molto simili tra loro, ma senza garanzie teoriche che lo testimoniano. É, infatti, sempre possibile iniziare con un allineamento pairwise incompatibile con i successivi allineamenti, per poi propagarlo nelle iterazioni successive: tale fenomeno causa un allineamento multiplo non ottimo.

Con questa tipologia di algoritmi è necessario solitamente compiere $\binom{N}{2} \in O(N^2)$ allineamenti pairwise, dunque la complessità del problema dipende da quella della tecnica e dell'algoritmo usato per l'allineamento pairwise.

2.3 Metodi alignment-free

I metodi basati sugli allineamenti presentano alcune problematiche, tra le quali il proibitivo costo computazionale per sequenze lunghe come quelle genomiche. Ciò ha giustificato la necessità di approdare a metodi che non siano basati sugli allineamenti, detti *alignment-free*. Il loro successo è conseguenza del fatto che costituiscono una valida alternativa agli allineamenti, dati gli ottimi risultati che sono in grado di ottenere con un costo computazionale generalmente moderato, specie considerando l'odierno ammontare di dati disponibili [51].

Esistono diversi approcci di tipo alignment-free, di seguito ne presentiamo alcuni basati sul conteggio di k-mer nelle sequenze, al fine di produrre statistiche usate come metriche. L'idea è che il grado di similarità tra due sequenze è misurabile confrontando le frequenze con cui appaiono i k-mer nelle sequenze stesse. É stato dimostrato, infatti, che una sequenza può essere rappresentata dalla distribuzione delle sue parole affinché sia possibile effettuare un confronto alignment-free efficace [31].

Uno dei metodi più semplici che si propone di evidenziare la correlazione tra due sequenze è D_2. Siano $X \in \Omega^{n>0}$ e $Y \in \Omega^{m>0}$ due sequenze. Dato un k-mer $w \in \Omega^{k \leq \min\{n,m\}}$, sia

$$X_w = \sum_{i=1}^{n-k+1} \mathbb{1}_{[X_i=w_1,\ldots,X_{i+k-1}=w_k]}$$

il conteggio delle occorrenze di w in X, dove $\mathbb{1}_{[condizione]}$ è la funzione indicatore, uguale a 1 se la condizione è vera, 0 altrimenti.

Si può definire quindi D_2, basandosi sui contatori X_w e Y_w per le sequenze X e Y.

Definizione 1 (D_2)**.**

$$D_2 = \sum_{w \in \Omega^k} X_w Y_w \tag{2.3}$$

Ciò che ha portato alla definizione di D_2 è l'idea che X_w e Y_w sono simili quanto più relazionate sono le sequenze X e Y. Sebbene D_2 sia veloce da calcolare, è stato dimostrato che non è adatta allo scopo per cui è stata ideata [43]. Nonostante questo, D_2 è la base sulla quale sono costruite tutte le statistiche trattate in questo lavoro.

Alcune varianti, come D_2^z e D_2^S, si propongono di risolvere alcuni problemi legati a falsi positivi, dove la statistica di una sequenza intacca l'altra introducendo dei *bias* indesiderati.

Definizione 2 (D_2 standardizzata)**.**

$$D_2^z = \frac{D_2 - \mathbb{E}[D_2]}{\sqrt{Var[D_2]}}$$

Definizione 3 (Shepp D_2)**.**

$$D_2^S = \sum_{w \in \Omega^k} \frac{(X_w - \mathbb{E}[XY_w])(Y_w - \mathbb{E}[XY_w])}{\sqrt{(X_w - \mathbb{E}[XY_w])^2 + (Y_w - \mathbb{E}[XY_w])^2}}$$

dove la S sta per *Shepp*, o *self-standardized*, mentre XY_w indica il numero di occorrenze di w nella sequenza ottenuta concatenando X e Y.

Tra le varianti di D_2, quella che ottiene risultati generalmente migliori è D_2^*.

Definizione 4 (D_2^*)**.**

$$D_2^* = \sum_{w \in \Omega^k} \frac{(X_w - \mathbb{E}[XY_w])(Y_w - \mathbb{E}[XY_w])}{Var[XY_w]}. \tag{2.4}$$

2.3.1 Profili entropici

Il grosso svantaggio delle statistiche presentate è che sono legate alla scelta di un valore fissato per k, il quale influenza le performance in modo cruciale, ma che non può essere noto a priori.

Per risolvere questo problema è si introducono i *profili entropici*, basati anch'essi sul conteggio di k-mer ma che utilizzano valori di k variabile. A partire da essi è possibile, infatti, estendere le statistiche appena introdotte. Come già detto, un confronto alignment-free efficace può partire dalla distribuzione dei k-mer in una sequenza. Partiamo proprio da qui per costruirne il profilo entropico.

CGR

La *chaos-game representation* (CGR) mappa una sequenza di DNA $X \in \Omega^{n>0}$ in una poligonale nello spazio continuo $\mathbb{R}^2$. Si consideri un quadrato unitario e si assegni a ciascuno dei suoi vertici un simbolo di Ω. A partire da un punto casuale nel quadrato, $CGR(X)$ associa a ogni simbolo $X_{i\in[1,n]}$ della sequenza un punto $x_{i\in[1,n]}$ nel quadrato:

$$\begin{cases} x_0 \sim \mathcal{U}(0,1)^2 \\ x_i = x_{i-1} + \frac{1}{2}(y_i - x_{i-1}) \ i \in [1,n] \end{cases} \tag{2.5}$$

dove

$$y_i = \begin{cases} (0,0) & \text{se } X_i = A \\ (0,1) & \text{se } X_i = C \\ (1,0) & \text{se } X_i = G \\ (1,1) & \text{se } X_i = T \end{cases}$$

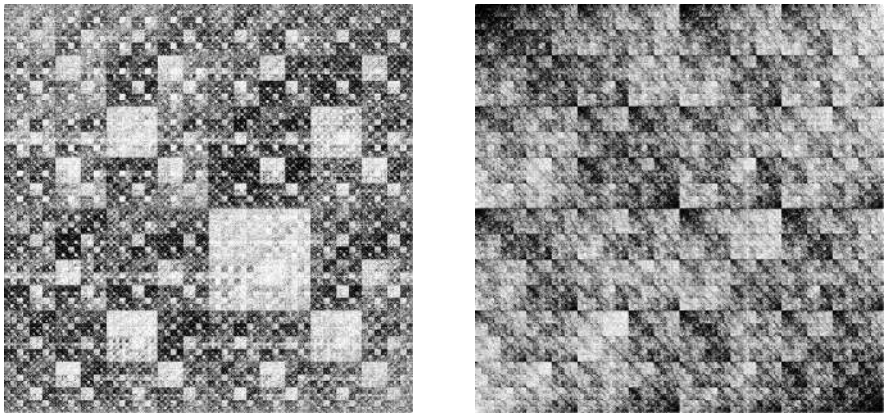


Figura 2.5: CGR di sequenze di *Synechococcus sp* (a sinistra) e *Bacillus subtilis* (a destra) [2]. Per la CGR di una sequenza di SARS-CoV-2 e un semplice script per la visualizzazione, si veda la sezione B.2 in appendice.

In alcuni casi, il fatto che, per esempio, $d(A, C) \neq d(A, T)$ (dove in questo caso per d s'intende la distanza euclidea), potrebbe costituire un problema, perciò è stata introdotta la *universal sequence map* (USM) [25], che mappa una sequenza $X \in \Sigma^*$ in una poligonale nello spazio $\mathbb{R}^{\|\Sigma\|}$. La USM di una sequenza genomica $X \in \Omega^{n>0}$ è definita in modo analogo alla CGR: sia assegnato un vettore della base canonica di $\mathbb{R}^4$ a ciascun simbolo dell'alfabeto, che si troverà ora sul vertice di un ipercubo unitario. Il resto è immediato:

$$\begin{cases} x_0 \sim \mathcal{U}(0,1)^4 \\ x_i = x_{i-1} + \frac{1}{2}(y_i - x_{i-1}) \ i \in [1, n] \end{cases} \quad y_i = \begin{cases} (1,0,0,0) & \text{se } X_i = A \\ (0,1,0,0) & \text{se } X_i = C \\ (0,0,1,0) & \text{se } X_i = G \\ (0,0,0,1) & \text{se } X_i = T \end{cases}$$

Per comodità, tuttavia, ci riferiremo sempre alla CGR. Vale la seguente proprietà.

Proprietà 1 (CGR suffix property)**.** Data la CGR di una sequenza $X \in \Omega^{n>0}$, ogni sottostringa di X con un prefisso comune di lunghezza k, si trova nello stesso sottoquadrato di lato 2^{-k}.

Se un motivo è frequente, quindi, il quadrato che lo rappresenta è più densamente popolato di punti. Questo, tra le altre cose, permette di distinguere le sequenze generate artificialmente, che tendono a occupare tutto il quadrato, da quelle naturali che, contenendo informazione, non hanno una distribuzione uniforme.

Dividendo il quadrato k volte in 4^k sottoquadrati di area 4^{-k}, si possono contare i punti presenti in un sottoquadrato, corrispondente a un preciso suffisso, per poi costruire un istogramma, che conta i k-mer nella sequenza

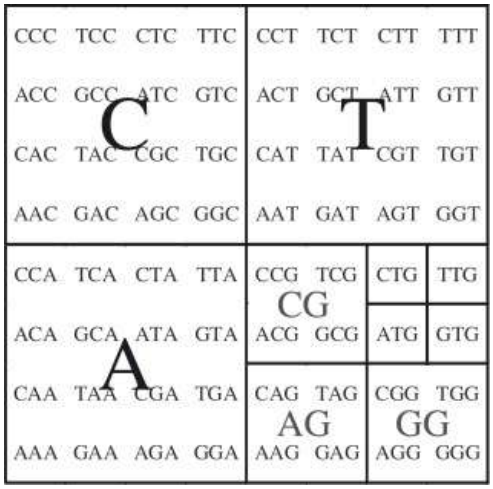


Figura 2.6: Rappresentazione di quanto enunciato nella proprietà 1. Sottosequenze che terminano con lo stesso suffisso corrispondono a punti all'interno dello stesso sottoquadrato, etichettato con tale suffisso [24].

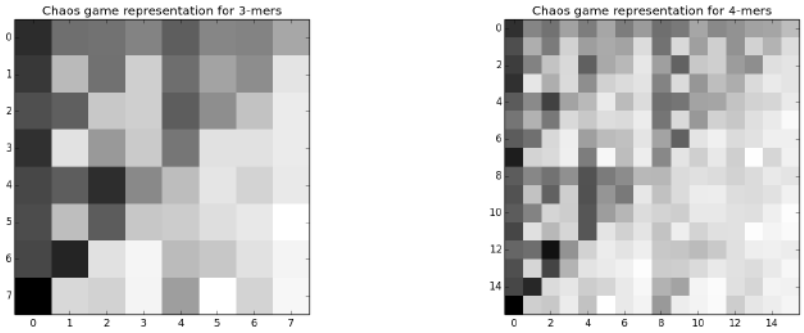


Figura 2.7: Data la stessa CGR, con risoluzioni diverse cambia la precisione nel conteggio dei suffissi [3]. Siccome la conoscenza di un punto è determinata dal solo punto precedente, la CGR generalizza i modelli di Markov di ordine k, dato che a ciascun punto è associata una sottosequenza di lunghezza k.

mappata. Si noti come la scelta di k sia determinante: è anche chiamato *risoluzione*, dato che all'aumentare di k aumenta la precisione dell'istogramma e la qualità dei campioni.

Dall'istogramma costruito a partire dalla CGR, quindi, è possibile stimare la distribuzione dei k-mer.

Finestra di Parzen-Rosenblatt

Il metodo della finestra di Parzen-Rosenblatt è utile per stimare la funzione di distribuzione di una variabile aleatoria. Siano $a = (a_1, \ldots, a_n)$ dei campioni iid prelevati da una distribuzione f non nota. La stima della

distribuzione ottenuta dal campionamento a è

$$\hat{f}_h(x;a) = \frac{1}{n}\sum_{i=1}^{n}\kappa_h(x-a_i) = \frac{1}{nh}\sum_{i=1}^{n}\kappa\left(\frac{x-a_i}{h}\right) \tag{2.6}$$

dove h è un parametro chiamato *bandwidth* (la larghezza della *finestra*) che influisce particolarmente sulle performance della stima ed è determinato in modo da minimizzare un certo errore, ma che avendo a che fare con ipercubi di lato unitario è possibile porre $h = 1$ senza indagare ulteriormente; mentre κ è il kernel, una funzione non negativa che rispetta alcune proprietà, come la seguente.

Proprietà 2.

$$\int \kappa(x)dx = 1 \tag{2.7}$$

Facendo la combinazione lineare dei kernel centrati sui campioni raccolti, $\hat{f}$ stima f. Se la funzione kernel è differenziabile, questa procedura equivale a *levigare* la distribuzione empirica di partenza, cioè l'istogramma, mantenendo le proprietà di una funzione di distribuzione:

$$\int \hat{f}(x;a)dx = 1.$$

In [24], per esempio, si adotta un kernel gaussiano, il che spiega il bisogno di definire la USM, preferita alla CRM in quel caso. Con le proprietà che comporta questa scelta, alcuni calcoli si semplificano, per giungere alla definizione dell'*entropia quadratica di Rényi*, sulla quale questo lavoro non intende soffermarsi.

Kernel

S'introduce, ora, un kernel basato sulla geometria della CGR, in modo da arrivare a una ridefinizione delle statistiche basate sui conteggi che non considerino un k fissato.

Si consideri $CGR(X) \subset [0,1] \times [0,1] \subset \mathbb{R}^2$. Dato l'intervallo unidimensionale

$$\mathcal{A}_{k,x_j} = (2^{-k}\lfloor 2^k x_j \rfloor, 2^{-k}\lfloor 2^k x_j \rfloor + 2^k) \tag{2.8}$$

di ampiezza $V(\mathcal{A}_{k,x_j}) = 2^{-k}$, sia

$$\mathcal{I}_{k,x_j}(x) = \mathbb{1}_{[x\in\mathcal{A}_{k,x_j}]}. \tag{2.9}$$

In due dimensioni esso si estende a $x_j = \vec{x}_j = (x_j^{(1)}, x_j^{(2)})$ e $x = \vec{x} = (x^{(1)}, x^{(2)})$, perciò

$$\begin{aligned}\mathcal{I}_{k,(x_j^{(1)},x_j^{(2)})}(x^{(1)},x^{(2)}) &= \mathcal{I}_{k,x_j^{(1)}}(x^{(1)}) \cdot \mathcal{I}_{k,x_j^{(2)}}(x^{(2)}) \\ &= \begin{cases} 1 & \text{if } x^{(1)} \in \mathcal{A}_{k,x_j^{(1)}} \wedge x^{(2)} \in \mathcal{A}_{k,x_j^{(2)}} \\ 0 & \text{altrimenti} \end{cases}\end{aligned} \tag{2.10}$$

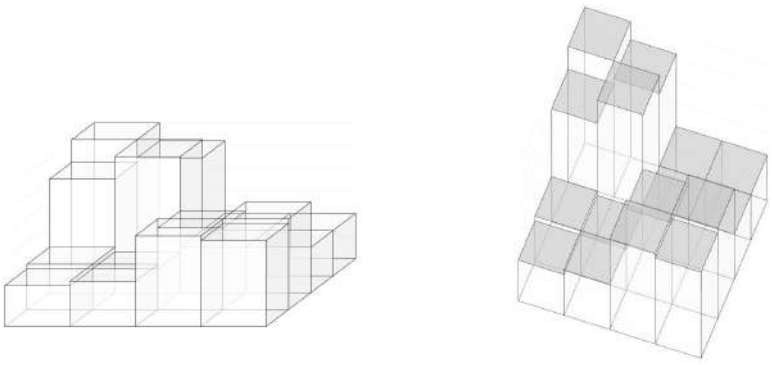


Figura 2.8: Rappresentazione semplificata di un istogramma costruito sulla base della stessa CGR con risoluzione $k = 2$, visto da due angolazioni diverse. L'altezza di ciascuna colonna è proporzionale alla densità di punti nel sottoquadrato alla base.

che definisce un sottoquadrato $\subseteq [0,1] \times [0,1]$ di area 4^{-k}. Ci riferiremo sempre e solo al caso bidimensionale, quindi indicheremo tale intervallo con la notazione più semplice introdotta in 2.8 e 2.9.

Il kernel che useremo è una combinazione lineare delle funzioni $\mathcal{I}_{k,x_j}(x)$,

$$\kappa_{L,x_j}(x) = \sum_{k=0}^{L} h_k \cdot \mathcal{I}_{k,x_j}(x) \tag{2.11}$$

dove h_k rappresenta l'altezza (o il peso, si veda l'immagine 2.7) dato a ciascun sottoquadrato considerato con $\mathcal{I}_{k,x_j}(x)$; mentre L è la *risoluzione di Markov*, un parametro che stabilisce il limite superiore per k, cioè k varia in $[1, L]$. Per la proprietà 2,

$$1 = \int \kappa_{L,x_j}(x)dx = \int \sum_{k=0}^{L} h_k \cdot \mathcal{I}_{k,x_j}(x)dx = \sum_{k=0}^{L} h_k \int \mathcal{I}_{k,x_j}(x)dx$$

e, dato che $\mathcal{I}_{k,x_j}(x)$ definisce un quadrato di area 4^{-k},

$$\int \mathcal{I}_{k,x_j}(x)dx = V(\mathcal{A}_{k,x_j^{(1)}}) \cdot V(\mathcal{A}_{k,x_j^{(2)}}) = 4^{-k}$$

si ottiene

$$\sum_{k=0}^{L} h_k \int \mathcal{I}_{k,x_j}(x)dx = \sum_{k=0}^{L} h_k 2^{-2k}. \tag{2.12}$$

Definiamo ora il parametro

$$\phi = \frac{V(\mathcal{A}_k)h_k}{V(\mathcal{A}_{k-1})h_{k-1}} = \frac{h_k}{4h_{k-1}} \tag{2.13}$$

che rappresenta il rapporto tra volumi in risoluzioni consecutive. Da questa definizione ricaviamo $h_k = 4\phi h_{k-1} = \cdots = (4\phi)^k h_0$. Riprendendo la 2.12,

$$1 = \sum_{k=0}^{L} h_k 2^{-2k} = \sum_{k=0}^{L} (4\phi)^k h_0 2^{-2k} = h_0 \sum_{k=0}^{L} \phi^k \Rightarrow h_0 = \frac{1}{\sum_{k=0}^{L} \phi^k}.$$

Giungiamo finalmente a una definizione più chiara del kernel 2.11,

$$\kappa_{L,\phi,x_j}(x) = \frac{\sum_{k=0}^{L}(4\phi)^k \mathcal{I}_{k,x_j}}{\sum_{k=0}^{L}\phi^k}$$

che rappresenta l'idea di pesare, con potenze di 4ϕ, ogni $\mathcal{I}_{k,x_j}$.

Ora è possibile tornare alla stima 2.6:

$$\begin{aligned}\hat{f}_{L,\phi}(x_i) &= \frac{1}{n}\sum_{j=1}^{n}\kappa_{L,\phi,x_j}(x_i) \\ &= \frac{1}{n}\sum_{j=1}^{n}\frac{\sum_{k=0}^{L}(4\phi)^k \mathcal{I}_{k,x_j}(x_i)}{\sum_{k=0}^{L}\phi^k} \\ &= \frac{\sum_{k=0}^{L}(4\phi)^k \sum_{j=1}^{n}\mathcal{I}_{k,x_j}(x_i)}{n\sum_{k=0}^{L}\phi^k}. \end{aligned} \tag{2.14}$$

Data la geometria della CGR, per l'equazione 2.10, si ha che $\mathcal{I}_{k,x_j}(x_i) = 1$ se il k-mer corrispondente alla coordinata x_i è equivalente al k-mer rappresentato alla coordinata x_j. Per questo motivo $\sum_{j=1}^{n}\mathcal{I}_{k,x_j}(x_i)$ si può calcolare semplicemente contando il numero di k-mer comuni lungo l'intera sequenza:

$$\sum_{j=1}^{n}\mathcal{I}_{k,x_j}(x_i) = c([i-k+1,i])$$

dove, se la sequenza in esame è $X = X_1 \dots X_n \in \Omega^{n>0}$, il numero di occorrenze del suffisso $X_{i-k+1}, \dots, X_i$ è indicato con $c([i-k+1,i])$. Infine, la formula diventa

$$\hat{f}_{L,\phi}(x_i) = \frac{1 + \frac{1}{n}\sum_{k=1}^{L} 4^k\phi^k \cdot c([i-k+1,i])}{\sum_{k=0}^{L}\phi^k} \tag{2.15}$$

con $L \geq 1$ e, si ricorda, n è la lunghezza della sequenza e i ne è l'i-esimo simbolo.

Risulta evidente la comodità indotta dall'utilizzo del kernel 2.11: per stimare la densità è sufficiente contare le sottostringhe di lunghezza $k \in [1, L]$, tenendo conto che spesso si preferiscono L bassi [26].

Per quanto riguarda l'influenza del parametro ϕ, siccome questo rappresenta il rapporto tra un peso a una risoluzione e un peso alla risoluzione precedente, per $\phi \to +\infty$ conta solo la risoluzione L

$$\lim_{\phi\to+\infty}\hat{f}_{L,\phi}(x_i) = \frac{4^L \cdot c([i-L+1,i])}{n}$$

mentre per $\phi \to 0^+$ si ha una distribuzione uniforme

$$\lim_{\phi\to 0^+}\hat{f}_{L,\phi}(x_i) = 1.$$

In effetti, $\hat{f}_{L,\phi}$ può essere interpretata come combinazione lineare del numero di suffissi fino a una certa lunghezza, con peso crescente se $\phi > 1/4$ o decrescente se $\phi < 1/4$.

Ottenuta la stima è possibile definire il profilo entropico.

Definizione 5 (Profilo entropico)**.** Data $\hat{f}_{L,\phi}$ si definisce, per ogni $i \in [1, n]$,

$$EP_{L,\phi}(i) = \frac{\hat{f}_{L,\phi}(x_i) - \mathbb{E}[\hat{f}_{L,\phi}(x_i)]}{\sqrt{Var[\hat{f}_{L,\phi}(x_i)]}} \tag{2.16}$$

funzione della posizione i, che misura le deviazioni della densità dalla sua aspettazione per ciascuna coordinata (o, equivalentemente, per ogni simbolo) di tutti i suffissi che compaiono nella sequenza originale. Il profilo entropico, quindi, riesce a catturare un certo tipo di rilevanza di una regione rispetto all'intero genoma.

2.3.2 EP_2 e EP_2^*

Possiamo generalizzare quanto ricavato in 2.15 definendo la *simple entropy*.

Definizione 6 (Simple entropy)**.** La simple entropy di una parola $w = w_1 \dots w_L$ di una sequenza $X \in \Omega^{n>0}$ è

$$X_{SE_w} = \frac{\sum_{k=1}^{L} a_k c_{w,k}}{\sum_{k=1}^{L} a_k} \tag{2.17}$$

dove $c_{w,k}$ è il numero di occorrenze del suffisso $w_{L-k+1} \dots w_L$ lungo l'intera sequenza X, mentre a_k sono dei pesi generici.

L'intuizione è che i profili entropici possono essere usati in sostituzione del semplice conteggio dei k-mer in modo da poter costruire statistiche alignment-free non basate su una lunghezza fissata, ma che ammettono più risoluzioni. Questo suggerisce che è possibile adattare le statistiche definite a inizio sezione, per esempio D_2 e D_2^*, sostituendo il contatore dei k-mer con la simple entropy.

Consideriamo due sequenze $X \in \Omega^{n>0}$ e $Y \in \Omega^{m>0}$ e le relative simple entropy X_{SE_w} e Y_{SE_w}.

Definizione 7 (EP_2)**.**

$$EP_2 = \sum_w X_{SE_w} Y_{SE_w} \tag{2.18}$$

Definizione 8 (EP_2^*)**.**

$$EP_2^* = \sum_w \frac{(X_{SE_w} - \mathbb{E}[XY_{SE_w}])(Y_{SE_w} - \mathbb{E}[XY_{SE_w}])}{Var[XY_{SE_w}]}. \tag{2.19}$$

Il calcolo di $\mathbb{E}[XY_{SE_w}]$ e $Var[XY_{SE_w}]$ è in [31]. Sia EP_2, sia EP_2^*, possono essere computati in tempo e spazio lineari.

2.3.3 Altre misure

Ci sono altre misure, come per esempio la *divergenza di Kullback-Leibler* o la *divergenza di Jensen–Shannon*, basate entrambe sul confronto di distribuzioni di probabilità, cioè la stessa idea che ha portato alla definizione di EP. Sarebbe quindi interessante ampliare il lavoro includendo anche i risultati prodotti da questi metodi rendendo più vario il confronto e testandone anche le prestazioni.

2.4 Alberi filogenetici

Un albero filogenetico è un diagramma che rappresenta la correlazione tra sequenze opportunamente confrontate, disponendole in accordo con la loro similarità reciproca. Il modo in cui l'albero si dirama suggerisce la storia dei cambiamenti di una data sequenza e i possibili punti in cui modifiche diverse sono intervenute sulla stessa sequenza. La distanza tra due sequenze è definita sulla base della posizione reciproca dei nodi assegnati a esse. Gli alberi filogenetici sono ampiamente utilizzati: dall'organizzazione evoluzionistica di specie, alla filodinamica di virus.

Formalmente, un albero filogenetico è un grafo non diretto e aciclico $T = (V, E)$, dove V è l'insieme dei nodi ed E è l'insieme dei rami. Siano $u, v \in V$ due nodi connessi da un ramo $e \in E$: tale ramo è definito come $e = (u, v) \in E$. Si noti come l'ordine sia irrilevante dato che il grafo è non diretto, quindi $(u, v) \equiv (v, u)$. L'insieme dei nodi terminali, cioè le foglie dell'albero, è definito come

$$I = \{v \in V : deg(v) = 1\} \subset V$$

dove la funzione grado, $deg(\cdot)$, indica il numero di rami incidenti nel nodo che ha come argomento. Si noti che $\deg(v) = 3\ \forall v \in V/I$. Nel caso di alberi filogenetici, alle foglie sono assegnate le sequenze in analisi (tanto che, di fatto, non c'è distinzione tra sequenze e nodo assegnato a essa), mentre i nodi interni rappresentano gli antenati comuni tra di esse; i rami e la loro disposizione determinano la relazione di antenati tra i nodi.

Gli alberi possono essere radicati o non: la radice, se presente, rappresenta in generale l'antenato comune a tutte le sequenze analizzate.

Gli alberi, inoltre, si dividono in *filogrammi* o *cladogrammi*, a seconda rispettivamente della presenza o meno di una proporzionalità tra la lunghezza (o peso) dei rami e la distanza degli elementi rappresentati ai nodi [28].

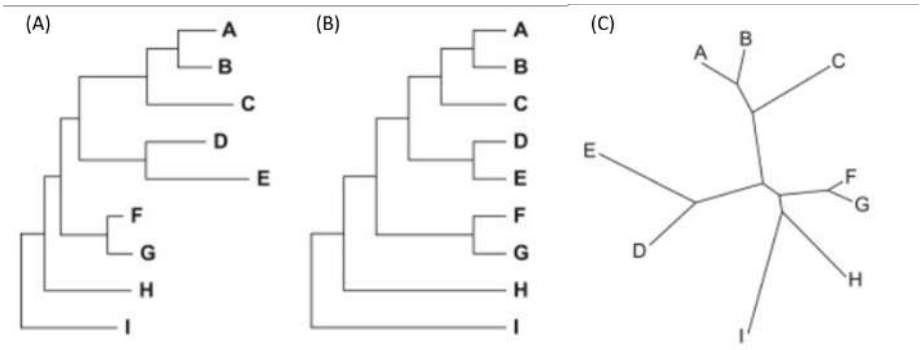


Figura 2.9: (A) Filogramma radicato (B) Cladogramma radicato (C) Filogramma non radicato [38].

2.4.1 Neighbor-Joining

Neighbor-Joining (NJ) è un algoritmo greedy utilizzato per costruire alberi filogenetici a partire da una matrice di distanze tra sequenze precedentemente confrontate.

Sia I l'insieme di sequenze confrontate. Si indica la generica distanza tra la i-esima sequenza e la j-esima con $d(i, j)$. NJ iterativamente seleziona una coppia di sequenze da I, costruisce un sottoalbero e agglomera la coppia di sequenze selezionate, in modo da diminuire di 1 la cardinalità di I. Le coppie di sequenze sono scelte in modo da minimizzare la seguente funzione obiettivo:

$$Q(i, j) = (N - 2)d(i, j) - \sum_{l=1}^{N} d(i, l) - \sum_{l=1}^{N} d(l, j)$$

dove N è la cardinalità di I e le somme ne scandiscono tutti gli elementi.

Siano f e g le sequenze selezionate alla generica iterazione. L'algoritmo calcola la lunghezza del branch (f, u) nel seguente modo:

$$d(f, u) = \frac{1}{2}d(f, g) + \frac{1}{2(N - 2)}\left[\sum_{l=1}^{N} d(f, l) - \sum_{l=1}^{N} d(g, l)\right]$$

e $d(g, u)$ è ottenuta per simmetria.

Infine, NJ sostituisce f e g con u nella matrice distanza, aggiornando le distanze che coinvolgono u con

$$d(u, l) = \frac{1}{2}[d(f, l) - d(f, u)] + \frac{1}{2}[d(g, l) - d(g, u)]$$

per ogni sequenza $l \in I/\{u\}$.

Questo algoritmo presenta una serie di vantaggi [33] che lo rendono il più noto e utilizzato, uno di questi è la possibilità di essere implementato in tempo $O(N^3)$.

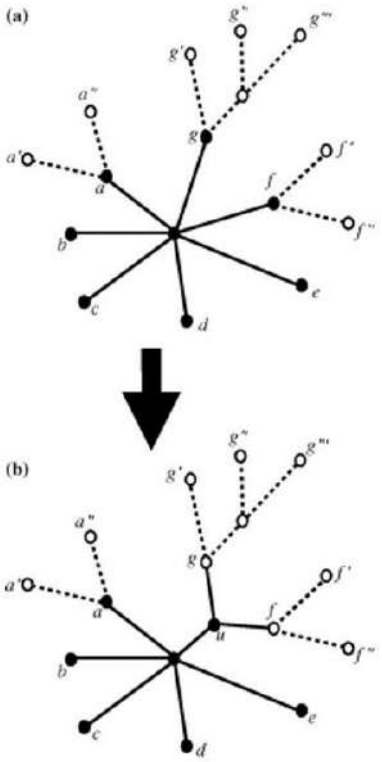


Figura 2.10: NJ agglomera le due sequenze f e $g \in I$ in un unico elemento $u \in I$, riducendo la cardinalità di I.

2.4.2 Altri metodi

Esistono altri metodi basati, come NJ, su una data matrice di distanze, per esempio *UPGMA* e *WPGMA*. Per questi è presente un breve approfondimento in B.3, poiché sono tipicamente sostituiti o affiancati a Neighbor-Joining nelle analisi come quella condotta in questo lavoro.

Gli altri due approcci più famosi sono *maximum likelihood*, il quale restituisce un albero che massimizza la probabilità di generare le sequenze osservate, e *maximum parsimony*, il quale identifica l'albero che richiede il minor numero di sostituzioni che spieghino le differenze osservate tra le sequenze.

2.4.3 Confrontare alberi

Per *misurare* quanto due alberi filogenetici siano *diversi* esistono diverse funzioni e una delle più semplici è la *distanza di Robinson-Foulds*.

Si consideri un albero filogenetico non radicato T. Rimuovendo un ramo da T, si ottengono due sottoalberi, che costituiscono la partizione indotta dal ramo rimosso. Si definisce *banale* una partizione indotta da un ramo che ha una foglia come estremo. Sia $S(T)$ l'insieme delle partizioni non banali di T.

Definizione 9 (Distanza di Robinson-Foulds non pesata)**.** Siano T_1 e T_2 alberi filogenetici non radicati, costruiti sullo stesso insieme di sequenze. Allora

$$d_{RF}(T_1, T_2) = |S(T_1) \setminus S(T_2)| + |S(T_2) \setminus S(T_1)|. \tag{2.20}$$

Essa rappresenta il numero di partizioni non banali non comuni tra i due alberi. Le partizioni banali sono escluse perché condivise per costruzione, dato che l'insieme di sequenze sulle quali sono costruiti gli alberi è lo stesso. Se ai rami è associato un peso, la variante pesata della distanza di Robinson-Foulds conta ciascuna partizione con il peso del ramo che l'ha indotta.

Un'altra funzione molto utilizzata è il *branch score*. Anziché confrontare la topologia degli alberi, come nel caso precedente, questa misura considera il diverso peso dei rami. Siano T_1 e T_2 due alberi filogenetici non radicati, costruiti sullo stesso insieme di sequenze. Il ramo e_1 di T_1 e il ramo e_2 di T_2 si dicono equivalenti se inducono due partizioni equivalenti (si noti che tutte le partizioni banali sono equivalenti). Il branch score si calcola sommando i valori assoluti delle differenze di peso tra rami equivalenti

$$d_{BS}(T_1, T_2) = \sum_{\substack{e_1 \in T_1, e_2 \in T_2 \\ \text{equivalenti}}} |w(e_1) - w(e_2)|, \tag{2.21}$$

dove se e_1 non ha un equivalente e_2, allora si considera $w(e_2) = 0$. Si noti che se tutti i rami hanno peso unitario, allora $d_{BS}(T_1, T_2) = d_{RF}(T_1, T_2)$.

Ci sono altri modi di calcolare il branch score, per esempio considerando il quadrato della differenza dei pesi, anziché il valore assoluto [41].

Un problema comune a queste funzioni è che non hanno un immediato significato statistico [47], per questo motivo ne esistono altre, basate su *nearest-neighbor interchange* o *subtree prune-and-regraft*.

2.5 Clustering

Il *clustering* è un problema che può essere formulato come segue. Dato un insieme di punti appartenenti a uno spazio metrico, si desidera trovare un raggruppamento di tali punti in sottoinsiemi, detti *cluster*, tali che

- punti nello stesso cluster siano *vicini* (o *simili*);
- punti di cluster diversi sono *distanti* (o *dissimili*).

In questo lavoro, i *punti* sono sequenze di DNA, quindi lo spazio metrico è (Ω^*, d), con d funzione distanza, incaricata di catturare la diversità delle sequenze. Tutte le distanze finora menzionate (euclidea, Hamming, edit) sono ammissibili. In generale, purché soddisfino certe proprietà, anche altre misure sono possibili, come quella indotta dall'utilizzo di EP_2 e EP_2^*.

Il clustering è un problema piuttosto generico, classificabile in molti modi diversi a seconda delle esigenze e delle applicazioni. Una possibile distinzione è la seguente, d'interesse per questo lavoro:

- il *k-clustering* cerca una partizione dell'insieme di input in k cluster, talvolta individuando dei punti rappresentativi per ogni cluster, minimizzando una certa funzione obiettivo;
- il *clustering gerarchico* dà maggiore attenzione al processo di partizione, con un numero di cluster variabile ed evidenziando le relazioni tra i cluster, rappresentate con un *dendrogramma*.

Di fatto, la costruzione di alberi filogenetici consiste in un clustering gerarchico, perché trova una partizione dei dati in input evidenziandone anche le relazioni e producendo un diagramma anziché un insieme di cluster disgiunti. In questo senso, il prodotto di un clustering gerarchico può essere convertito in un k-clustering con $k \in [1, N]$ dove N è il numero di sequenze in input. Facendo ciò, tuttavia, si perde tutta l'informazione relativa ai nodi interni dell'albero, che rappresentano le relazioni tra le sequenze in quanto antenati dei nodi foglia. Da un k-clustering ottenuto in questo modo, quindi, non è possibile risalire all'albero da cui è stato generato.

2.5.1 Clustering spettrale

NJ e UPGMA partono da una matrice di distanze per effettuare un clustering gerarchico. Nulla vieta, però, di partire dalla matrice per calcolare un clustering di diverso tipo.

Associando la matrice a un grafo, è possibile interpretare ciascuna sequenza come un nodo e ciascun ramo, che connette coppie di sequenze, ha una lunghezza indicata nella matrice. Il problema diventa quello di raggruppare sequenze con rami che le connettono molto corti nello stesso cluster, e fare in modo che i rami che connettono sequenze di cluster diversi siano molto lunghi. Questo prende il nome di *clustering spettrale*, ed è uno dei metodi più famosi di fare clustering a partire da una matrice. Il risultato prodotto, tuttavia, rischia di essere sbilanciato o poco rappresentativo, da qui la necessità di passare a un altro tipo di clustering. É presente un approfondimento sul clustering spettrale in appendice B.4.

2.5.2 Clustering da un albero filogenetico

Un altro modo per ottenere un clustering è partire da un albero filogenetico e *tagliarlo*. Il motivo di tale scelta è dovuto al fatto che ci sono diverse strategie per ottenere un clustering a partire dalle sequenze in input o dalla matrice di distanze, che non tengono conto del significato biologico. Partire da un albero filogenetico già prodotto significa partire da una struttura di cui si suppone il significato e che è stata ottenuta con un metodo apposito, prima di essere collassata a un k-clustering, perdendo informazione. In [27] si pone l'attenzione su questo fatto, aggiungendo che un clustering operato in questo modo trae vantaggio dall'albero catturando non solo la similarità tra le sequenze, ma anche le relazioni, contrariamente ai clustering che partono dalla sola matrice.

Il taglio dell'albero può risultare comunque non banale poiché si tratta, di fatto, di un altro problema di ottimizzazione. Riprendendo la definizione di albero filogenetico come grafo non diretto aciclico proposta nella sezione 2.4, siano u e v due foglie. La lunghezza del cammino tra di esse è indicata con $d(u, v)$ ed è la somma di tutti i contributi dei rami che compongono il cammino, indicati con w: se le due foglie sono connesse da un unico ramo $e = (u, v)$, allora $d(u, v) = w(u, v) = w(e)$.

Un modo naturale di definire un clustering delle foglie I è quello di associare un clustering a un *cutset* $C \subseteq E$ dei rami di T. Definiamo una partizione $\{L_1, \ldots, L_k\}$ di I un clustering *ammissibile* se è ottenuto rimuovendo l'insieme di rami C da E e assegnando alle foglie di ciascuna componente connessa rimasta un cluster $L_{i \in [1,k]}$. Si noti che valgono le seguenti proprietà:

- $k \leq \|C\| + 1$;

- $\bigcup_{i=1}^{k} L_i = I$;
- $L_i \cap L_j = \varnothing \ \forall i \neq j$.

Si definisce anche, per un dato albero T, la funzione $f_T : 2^I \to \mathbb{R}$ la quale mappa un sottoinsieme di I in un numero reale, usata come vincolo per definire la classe dei problemi di *min-cut partitioning*.

Definizione 10 (Min-cut partitioning problem)**.** Dato un albero $T = (V, E)$ con insieme delle foglie $I \subset E$ e un parametro $\alpha \in \mathbb{R}$, il min-cut partitioning problem richiede di trovare una partizione ammissibile $\{L_1, \ldots, L_k\}$ di I con k minimo, tale che $f_T(L_i) \leq \alpha$ per ogni $i \in [1, k]$.

La funzione f_T è scelta sulla base di alcune esigenze, introducendo allo stesso tempo degli svantaggi.

- Un modo naturale per limitare la diversità all'interno di un cluster è quello di vincolare tutte le distanze a coppie tra i membri dello stesso cluster a essere minori di α. Si pone quindi

$$f_T(L) = \max_{u,v \in L} d(u, v) \tag{2.22}$$

  ottenendo il *max-diameter* min-cut partitioning problem, il quale risulta però particolarmente sensibile alla presenza di *outlier*.

- La massima distanza tra punti dello stesso cluster potrebbe non essere rappresentativa della sua diversità, quindi è possibile ridefinire la funzione $f_T(L)$ come

$$f_T(L) = \sum_{(u,v) \in E(T|L)} w(u, v) \tag{2.23}$$

  dove $T|L$ è l'albero ristretto all'insieme di foglie L, cioè la componente connessa che induce il cluster L, mentre $E(T|L)$ ne considera i rami. Questo problema prende il nome di *sum-length* min-cut partitioning problem.

- Un altro problema che vale la pena citare è il *single-linkage* min-cut partitioning problem, che si ottiene con

$$f_T(L) = \max_{S \subset L} \{ \min_{u \in S, v \in L/S} d(u, v) \} \tag{2.24}$$

  il quale però produce un clustering molto più sensibile alla scelta di α rispetto ai precedenti.

Tutti i problemi appena definiti sono risolvibili in tempo lineare da varianti greedy dello stesso algoritmo [27].

2.5.3 Valutazione supervisionata

Supponendo di aver già prodotto un clustering, è possibile valutarne la qualità. Si possono eseguire due tipi di valutazione: supervisionata e *non* supervisionata. In questa sede sarà approfondita solo la prima di esse poichè è quella più affine al caso di studio presentato.

Sia $\{L_1, \ldots, L_k\}$ un clustering di I, dove ciascuna sequenza è etichettata con una certa classe. Per ogni cluster $L_{i\in[1,k]}$ e per ogni classe $c \in \mathcal{C}$, siano

- $m_{L_i} = \|L_i\|$;
- m_c = numero di sequenze della classe c;
- $m_{L_i,c}$ = numero di sequenze della classe c in L_i.

Definizione 11 (Entropia di un cluster)**.** L'entropia del cluster L_i è definita come

$$\varepsilon(L_i) = -\sum_{c\in\mathcal{C}} \frac{m_{L_i,c}}{m_{L_i}} \log_2 \frac{m_{L_i,c}}{m_{L_i}}. \tag{2.25}$$

L'entropia fornisce un'idea delle *impurità* presenti in un cluster, valutando le classi delle sequenze che vi appartengono. Vale sempre $\varepsilon(L_i) \in [0, \log_2 \|\mathcal{C}\|]$, dove, quando $\varepsilon(L_i) = 0$ tutte le sequenze di L_i appartengono alla stessa classe, mentre quando $\varepsilon(L_i) = \log_2 \|\mathcal{C}\|$ tutte le classi sono ugualmente presenti in L_i.

Tutto ciò, però, non è sufficiente, perché alcuni cluster possono essere ben valutati, mentre altri avere un'alta entropia, quindi è necessario misurare anche il livello di impurità contenuto nelle singole classi per una completa valutazione.

Definizione 12 (Entropia di una classe)**.** L'entropia della classe $c \in \mathcal{C}$ è definita come

$$\varepsilon(c) = -\sum_{L_i\in\{L_1,\ldots,L_k\}} \frac{m_{L_i,c}}{m_c} \log_2 \frac{m_{L_i,c}}{m_c}. \tag{2.26}$$

Tale valore misura quanto le sequenze della classe $c \in \mathcal{C}$ siano sparse tra i cluster, in un range che va da 0 (tutte le sequenze della classe c sono nello stesso cluster) a $\log_2 k$ (le sequenze della stessa classe sono ugualmente presenti in tutti i cluster).

Capitolo 3

Software utilizzati

In questo capitolo è proposta una breve ma dettagliata panoramica dei pacchetti e dei software utilizzati per le analisi, descrivendo, per ciascuno, le computazioni eseguite e i possibili parametri da variare, sottolineandone i valori assegnati.

3.1 Biopython

Biopython [29] è una vasta libreria di programmi per la bioinformatica scritti in linguaggio Python 3 (versioni precedenti supportano anche Python 2). A questa appartengono le classi SeqIO e AlignIO per la lettura o scrittura di sequenze genomiche e allineamenti in una grande varietà di formati supportati. Essa contiene nativamente anche dei tool per effettuare allineamento multiplo, pairwise (globale o locale), per costruire alberi filogenetici e eseguire clustering. Biopython include anche alcune classi per eseguire delle query su BLAST.

Una feature interessante è quella dei *command-line wrapper*: classi che consentono a un programma Python che ne fa uso di interfacciarsi con il terminale del sistema operativo e, tramite esso, eseguire altri software quali ad esempio ClustalW, MUSCLE, MAFFT, TCoffe e i tool delle suite EMBOSS e EMBASSY. Questa funzionalità consente, quindi, anche di fare uso del multithreading, lanciando più istanze in parallelo di tool diversi. Un tutorial per l'installazione e l'utilizzo è disponibile presso [4].

3.2 EMBOSS e EMBASSY

EMBOSS (*The European Molecular Biology Open Software Suite*) contiene al suo interno una vasta gamma di tool per la bioinformatica [44]. É utilizzato in ambito commerciale e accademico per diversi scopi, come l'allineamento di sequenze (tramite `needle`, `water`, `stretcher`), la ricerca di

fattori di trascrizione in sequenze di DNA (`tfscan`) e la ricerca all'interno di una sequenza tramite una matrice o un profilo (`profit`).

Per l'albero filogenetico ottenuto tramite allineamento pairwise in 4.2.1, è stato utilizzato il tool `stretcher` con i parametri standard: esso è basato sull'algoritmo di allineamento globale in spazio lineare di Myers e Miller ([40]) che usa la strategia citata nel paragrafo 2.2.3. Tutte le informazioni su EMBOSS, incluse le istruzioni di installazione, sono reperibili presso [5]. I dettagli sull'utilizzo di stretcher si trovano presso [6].

EMBASSY è anch'essa una suite di tool dall'interfaccia simile a EMBOSS. Fra i pacchetti disponibili, il pacchetto PHYLIPNEW utilizzato in questo lavoro consente di utilizzare da riga di comando i tool di Phylip secondo una sintassi simile ai comandi EMBOSS. EMBASSY è disponibile presso [7], mentre una descrizione delle funzioni di PHYLIPNEW è consultabile alla pagina [8].

3.3 ClustalW2

ClustalW2 è un tool appartenente al pacchetto Clustal dalla *Science Foundation Ireland*, utilizzato per l'allineamento multiplo di sequenze. Esso è la versione aggiornata, dal punto di vista dell'efficienza e dell'accuratezza, del software ClustalW, terza generazione dei software della classe Clustal.

La descrizione dell'algoritmo è disponibile presso la documentazione ufficiale [9].

1. Nel primo passo, le sequenze fornite in ingresso con un file in formato `.fasta` sono allineate a coppie tra loro. In questo modo, è possibile definire la distanza tra ogni possibile coppia di sequenze: è necessario eseguire quindi $N(N-1)/2$ allineamenti pairwise, dove N è il numero di sequenze fornite in input. Per questo motivo, è possibile utilizzare due diversi algoritmi:

   - il primo, più lento ma molto accurato, è l'algoritmo di programmazione dinamica per l'allineamento globale pairwise di Needleman-Wunsch, descritto in 2.2.1;
   - il secondo algoritmo, veloce e approssimato, è il metodo di Wilbur e Lipman 2.2.2, scelto quando la complessità del primo lo rende ineseguibile.

   La distanza calcolata in questo modo può essere visualizzata in output durante l'esecuzione, per ogni coppia di sequenze. É possibile inoltre scegliere se visualizzare i valori di score in forma percentuale, un valore tra 0 e 100 che indica la percentuale di similarità tra le due sequenze, oppure in forma assoluta, visualizzando l'effettivo score dell'allineamento.

2. Nel secondo passo, dalla matrice di distanze, ClustalW2 costruisce un albero guida con il metodo NJ, in cui le singole sequenze sono posizionate alle foglie e la lunghezza dei rami è proporzionale alla dissimilarità delle sequenze che essi collegano.

3. Il terzo passo di computazione è la realizzazione dell'algoritmo di allineamento progressivo che adotta il tree approach, descritto in 2.2.4: partendo dall'albero guida creato al punto precedente, costruisce l'allineamento globale multiplo delle sequenze in input.

ClustalW2 esegue, quindi, un algoritmo di allineamento progressivo basato su un approccio euristico: non è dunque garantito che esso restituisca il miglior allineamento possibile tra le sequenze.

L'esecuzione di questo tool dipende da una serie di parametri [10], che in questo lavoro sono stati lasciati ai valori di default. É possibile inoltre scegliere il formato del file in output in funzione delle successive esecuzioni da compiere. Per questo lavoro è stato scelto il formato `.phy` per facilitare il passaggio alle computazioni successive.

3.4 EP-sim

Il programma usato per calcolare EP_2 e EP_2^* si chiama `ep_sim` [30], e li calcola in tempo e spazio lineari. É scritto in linguaggio C++ e utilizza la libreria SeqAn, che include efficienti primitive per l'analisi di sequenze, in particolare biologiche [11].

Esso richiede due argomenti e cinque opzioni:

1. il nome del file contenente tutte le sequenze da confrontare, in formato `.fasta`;

2. il nome del file output che conterrà la matrice di score, con i punteggi del confronto tra tutte le possibili coppie di sequenze;

3. `L`, la risoluzione di Markov tale che $k \in [1, L]$ (di default pari a 5);

4. nella definizione di simple entropy 2.17 sono presenti dei pesi generici, che in `ep_sim` sono gaussiani:

$$a_k = e^{-\frac{(L-k)^2}{2\sigma^2}}$$

   con la deviazione standard $\sigma = 0.5$ di default, ma modificabile con `P`;

5. `B`, il background, di default `M1` cioè un modello markoviano di ordine 1;

6. `D`, la statistica selezionata. Con `D2`, che è di default, si seleziona EP_2, mentre con `D2*` si seleziona EP_2^*. Le analisi sono state condotte con entrambi i parametri, lasciando tutti gli altri invariati ai valori di default;

7. infine `R`, un flag che tiene conto del calcolo del *reverse complement*, `false` di default e lasciato tale.

3.5 Phylip

Phylip (*PHYLogeny Inference Package*) [47] è un pacchetto open-source pensato per l'analisi filogenetica. É costituito da 35 programmi, di cui sono disponibili il codice sorgente scritto in linguaggio C e anche l'eseguibile precompilato, completi di documentazione. Di questi, quelli che si sono dimostrati utili per l'analisi sono descritti nel seguito.

3.5.1 DNAdist

Si tratta di un modulo di Phylip che riceve in input un file di allineamento e restituisce una matrice di distanze in un file chiamato `outfile`. Il nome del file in input, se diverso da `infile`, va specificato. La computazione di dipende dal tipo di matrice di sostituzione nucleotidica selezionata fra le quattro disponibili: *Jukes-Cantor*, *Kimura*, *F84* e *LogDet*. Tutti gli altri parametri sono lasciati ai valori di default.

3.5.2 Neighbor

Si tratta dell'implementazione di NJ, descritto in 2.4.1, e di UPGMA. Accetta una matrice di distanze in input, il cui nome, se diverso da `infile`, va specificato.

I parametri del programma, riguardanti l'algoritmo da usare, il formato della matrice, il radicamento dell'albero, sono tutti lasciati ai valori di default.

L'output è scritto nel file `outfile` mentre l'albero, in formato Newick, nel file `outtree`. Il formato Newick è un modo di rappresentare alberi tramite parentesi e virgole, racchiudendovi le etichette associate ai nodi e le lunghezze dei rami. É popolare perché particolarmente comodo, infatti tutti i tool utilizzati in questo lavoro che producono o accettano alberi filogenetici fanno uso di questo formato.

3.5.3 Treedist

Calcola la distanza di Robinson-Foulds, qui chiamata anche *Symmetric Difference* oppure il branch score, entrambi descritti in 2.4.3. Il programma

riceve in input degli alberi in formato Newick nello stesso file `intree` e calcola la distanza secondo i parametri specificati:

- di default calcola il branch score, con il parametro `D` è possibile selezionare la symmetric difference, dato che gli alberi prodotti fino a questo momento non hanno pesi sui rami;
- con il parametro `P` del sottomenù `2` si richiede che il confronto avvenga tra tutte le coppie di alberi, in modo da confrontarli tutti tra loro;
- l'output è scritto in `outfile`, e con il parametro `F` si ottiene la matrice di distanze, mostrata in figura 4.6;
- tutti gli altri parametri sono lasciati ai valori di default.

3.6 TreeCluster

TreeCluster [12] è un programma scritto in linguaggio Python che implementa diversi algoritmi per risolvere altrettanti problemi della classe min-cut partitioning problem, definita nella definizione 10. Dato un albero in formato Newick e un parametro t, il programma restituisce il numero minimo di cluster sulle foglie dell'albero che rispetta alcuni vincoli, scelti anch'essi dall'utente. I parametri richiesti dal programma sono i seguenti:

- `i` per l'input;
- `t` la soglia t;
- `s` *branch support threshold*, non specificato;
- `m` specifica il metodo da utilizzare tra i seguenti:
  1. `max`
  2. `max_clade`
  3. `sum_branch`
  4. `sum_branch_clade`
  5. `avg_clade`
  6. `med_clade`
  7. `single_linkage`
  8. `single_linkage_cut`
  9. `single_linkage_union`
  10. `length`
  11. `length_clade`

12. `root_dist`
13. `leaf_dist_max`
14. `leaf_dist_min`
15. `leaf_dist_avg`

    La maggior parte di essi, ma non tutti, sono implementati con algoritmi $O(N)$, dove N è il numero di foglie dell'albero.

- `tf`, che se specificato a `argmax_clusters` trova e restituisce il valore della soglia $t^* \in [0, t]$ per il quale il numero di *singleton*, cioè i cluster contenenti un solo elemento, è minimizzato.

Il programma restituisce la lista delle sequenze, ciascuna delle quali affiancata a un indice che ne rappresenta il cluster di appartenenza. L'indice `-1` significa che la sequenza è in un cluster singleton.

In questa sede non sono specificati i valori assegnati ai parametri elencati, saranno ripresi successivamente.

3.7 Iroki

Per visualizzare gli alberi si è dimostrato particolarmente utile il software online Iroki [39]. Esso riceve un albero in formato Newick e un file che specifica i colori da assegnare ai rami a seconda delle sequenze che contengono. Tra le molteplici opzioni di visualizzazione e di output, le più importanti sono:

- `layout shape`, che può essere `rectangular`, `circular` o `radial`;
- per le prime due si può specificare anche `branch`, che può essere `true length` oppure `cladogram`;
- altre opzioni riguardano la presenza o meno delle etichette alle foglie, l'ordine di visualizzazione delle sequenze, la grandezza dell'albero, la risoluzione dell'immagine.

I parametri utilizzati non sono riportati in questa sede ma dove il tool sarà utilizzato per l'analisi, in quanto saranno usati diversi set di valori.

Capitolo 4

Dati e discussione

In questo capitolo è descritto il campionamento delle sequenze usate per l'analisi, le informazioni ritenute rilevanti e alcuni passaggi critici. Per ciascuno dei metodi in esame sono presentati i passi eseguiti, che permettono di ottenere l'albero filogenetico, con la speranza di proporre una chiara pipeline ripetibile anche su sequenze di diversa natura. Sono infine confrontati gli alberi ottenuti, con le diverse tecniche presentate nei precedenti capitoli.

4.1 Campionamento

Le sequenze di SARS-CoV-2 sono state scaricate dalla banca dati pubblica Gisaid [13], che ha permesso una rapida condivisione di dati su alcuni virus. All'interno di tale database, infatti, è possibile trovare più di 35 mila sequenze di virus, accompagnate da una lunga serie di parametri importanti, come la data, il luogo, la modalità ed eventuali commenti sul sequenziamento, i dati del paziente dal quale è stato prelevato il campione e altri. Un parametro importante è la *copertura*. Certi caratteri sequenziati possono non essere stati letti correttamente e, per tale motivo, si indicano con il simbolo `N`. La copertura è la percentuale di caratteri `N` all'interno della sequenza.

É possibile interrogare il database richiedendo che le sequenze rispettino le caratteristiche volute, variando le informazioni appena citate: In questo modo si può realizzare un campionamento efficace e mirato.

Per questo lavoro sono state selezionate sequenze in numero limitato, rispetto ai consueti numeri della ricerca e dell'analisi statistica. Questo è avvenuto in accordo con considerevoli risorse di calcolo richieste dai programmi per portare a termine la computazione, in particolar modo dagli allineamenti. Per mettere un'analisi trattabile dal punto di vista temporale, ma che fornisse risultati un minimo rilevanti statisticamente, abbiamo scelto cento sequenze, uniformemente da dieci diversi paesi: Cina, Colombia, Francia, Germania, Giappone, Italia, Regno Unito, Russia, Spagna e Stati

Uniti d'America. Per ciascuno stato sono state considerate dieci sequenze, con la speranza di poter ottenere dei risultati bilanciati e confrontabili dal punto di vista geografico. Ulteriori motivazioni a questa decisione saranno chiarite in seguito. Oltre al filtro spaziale, la selezione è avvenuta in accordo con i seguenti criteri:

- tutte le sequenze sono lunghe circa 30 *kb*;
- buona distribuzione temporale, le sequenze scelte sono distribuite quanto più uniformemente anche dal punto di vista della data specificata tra i metadati su Gisaid (fino al 24 Aprile 2020);
- tutte le sequenze provengono da campioni prelevati su pazienti umani;
- tutte le sequenze hanno una copertura strettamente minore del 5%.

Insieme alla sequenze, è possibile scaricare anche la tabella di riconoscimento in formato `.xls` oppure i metadati dello stato dei pazienti in formato `.tvs`. In questo lavoro sono state scaricate le seguenti informazioni per una corretta analisi dei risultati, soprattutto dal punto di vista geografico e temporale:

- identificatore della sequenza, stringhe del tipo `EPI_ISL_XXXXXX`, dove `X` è una cifra.

  A causa dei vincoli sul formato di input in alcuni programmi, che richiedono etichette di massimo 10 caratteri, a tutti gli identificatori è stato rimosso il prefisso `EPI_`, poiché comune e perciò non fondamentale per la distinzione delle sequenze;
- luogo di prelievo dei campioni, ove possibile, indicato nel formato "nazione / regione / città";
- data di prelievo.

Tali informazioni sono riportate per ciascuna sequenza in appendice A. Nello stesso capitolo è, inoltre, riportata la tabella riassuntiva della convenzione adottata per la visualizzazione degli alberi, con la sigla e il colore utilizzati per ogni nazione.

4.2 Alberi prodotti

In seguito riportiamo le procedure utilizzate per ciascuno dei quattro metodi proposti, a partire dal confronto delle sequenze, fino ad arrivare alla visualizzazione e discussione degli alberi filogenetici prodotti. In questa sezione i parametri di Iroki sono comuni a tutti i metodi, e sono i seguenti:

- `layout shape rectangular`
- `branch cladogram`
- `sorted`
- `size 40x60`
- `no scale bar`
- `show leaf labels`, con `size` circa a 3/4
- tutti gli altri parametri lasciati di default.

Sono stati scelti sulla base della possibilità di distinguere chiaramente le etichette alle foglie per tutti i possibili alberi. La scelta di radicare l'albero, invece, è una comodità estetica, per cui sono presenti sia alberi radicati, in questa sezione, sia le versioni non radicate, in appendice C.1.

4.2.1 Albero prodotto dall'allineamento pairwise

L'albero prodotto dall'allineamento pairwise è in figura 4.1. La pipeline utilizzata, eccetto per lo step 5, è interamente disponibile presso la repository [14] con le istruzioni di installazione ed esecuzione del codice. Gli step del processo sono i seguenti:

1. ogni file contenente sequenze è analizzato, e per ogni sequenza in esso contenuta è creato un singolo file `.fasta` in preparazione allo step successivo. Questo perché le sequenze scaricate da Gisaid possono in generale trovarsi distribuite in più file;

2. allineamento pairwise di tutte le coppie di sequenze tramite `stretcher`. In questo step, grazie ai command-line wrappers di Biopython, sono eseguite in parallelo 7 istanze di `stretcher`. Questa accortezza riduce drasticamente il tempo di computazione a circa 75 minuti, su una macchina con processore Intel Core i7-9750H con 16 GB di RAM. Al termine è prodotto un file `.phy` per ogni coppia di sequenze;

3. per ciascuno di questi file, è stato eseguito, sempre tramite il supporto di Biopython, l'esecuzione di `dnadist` (3.5.1) implementato nella funzione `fdnadist` [15] di PHYLIPNEW. Dalle singole matrici di distanza 2×2 è stata ottenuta una matrice complessiva che raggruppa le distanze di tutte le possibili coppie;

4. la matrice è data in input a `neighbor` grazie alla funzione `fneighbor` di PHYLIPNEW [16];

5. Iroki.

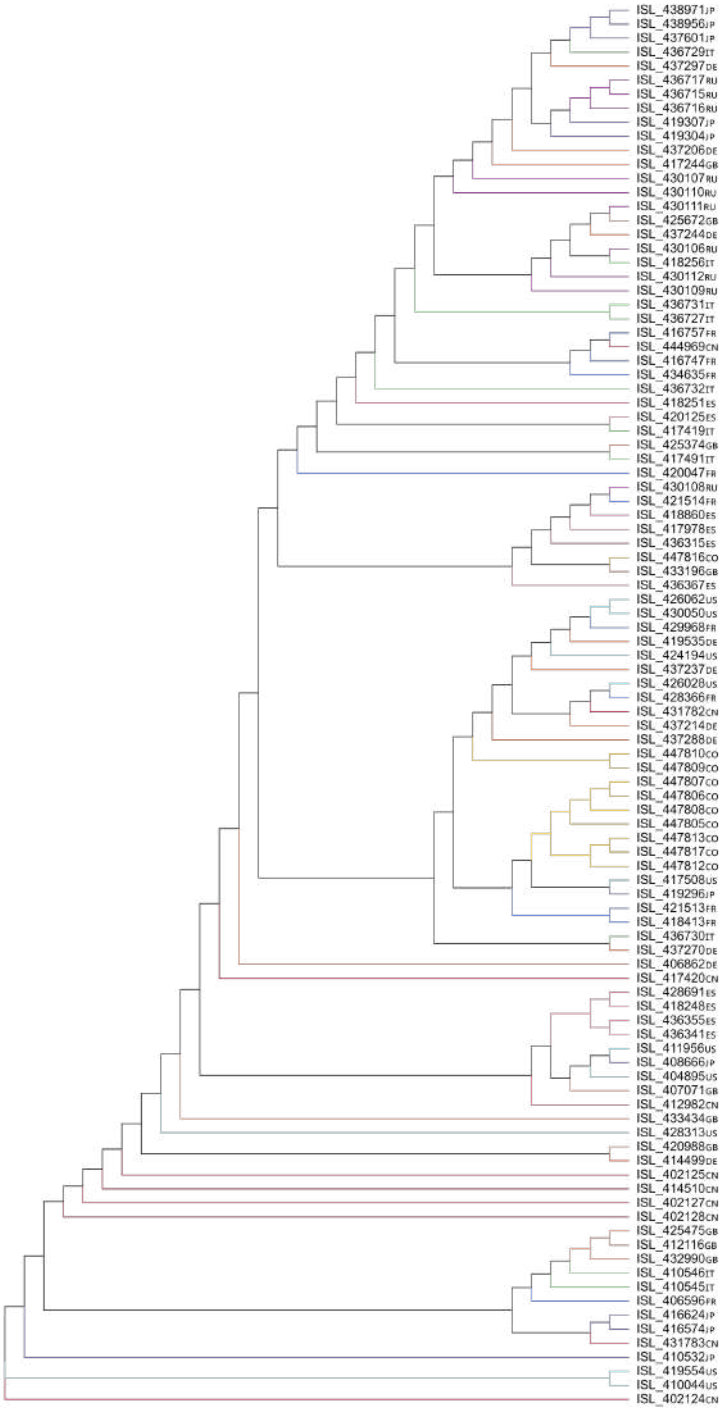


Figura 4.1: Albero prodotto da allineamento a coppie.

4.2.2 Albero prodotto dall'allineamento multiplo

L'albero prodotto dall'allineamento multiplo è in figura 4.2. L'analisi tramite allineamento multiplo è avvenuta partendo dalle sequenze scaricate da Gisaid e con l'esecuzione dei tool nel seguente ordine:

1. ClustalW2: fornendo in ingresso un file `.fasta` contenente tutte le sequenze, sono restituiti gli score dell'allineamento pairwise, la visualizzazione dell'allineamento progressivo insieme allo score del cluster di sequenze allineate e l'allineamento globale multiplo. Il tempo di esecuzione, a causa delle numerose operazioni compiute, è di circa cinque ore, tempo determinato su un laptop con processore Intel Core i7-8565U e 8 GB di memoria RAM.

   Il file `.aln` in output riporta un risultato, già a questo punto, interessante: su una lunghezza massima di 29903 caratteri, le sequenze ne condividono 28311, il che significa che le differenze tra di esse sono minime.

2. DNAdist, che ricevendo in input l'allineamento prodotto da ClustalW2 in formato `.phy`, restituisce, dopo pochi secondi, la distanza tra ogni coppia di sequenze;

3. `neighbor`, che riceve in input il file prodotto nel precedente step, e restituisce l'albero in formato Newick;

4. Iroki.

4.2.3 Alberi prodotti da EP_2 e EP_2^*

Gli alberi prodotti si trovano rispettivamente in figura 4.4 e 4.5. Partendo dalle sequenze in un unico file in formato `.fasta`, i tool utilizzati sono stati i seguenti:

1. EP-sim. Con i parametri elencati nella sezione 3.4, il tempo medio speso per l'intera computazione di EP_2 ed EP_2^* è, rispettivamente, di 4.199 e 5.629 secondi, misurato su un laptop con processore Intel Core i5-6300U e 4 GB di memoria RAM. La computazione è divisa in cinque fasi:

   (a) lettura delle sequenze in input;

   (b) calcolo delle entropie e del background;

   (c) calcolo di aspettazioni e varianze;

   (d) reverse complement;

   (e) produzione della matrice di score.

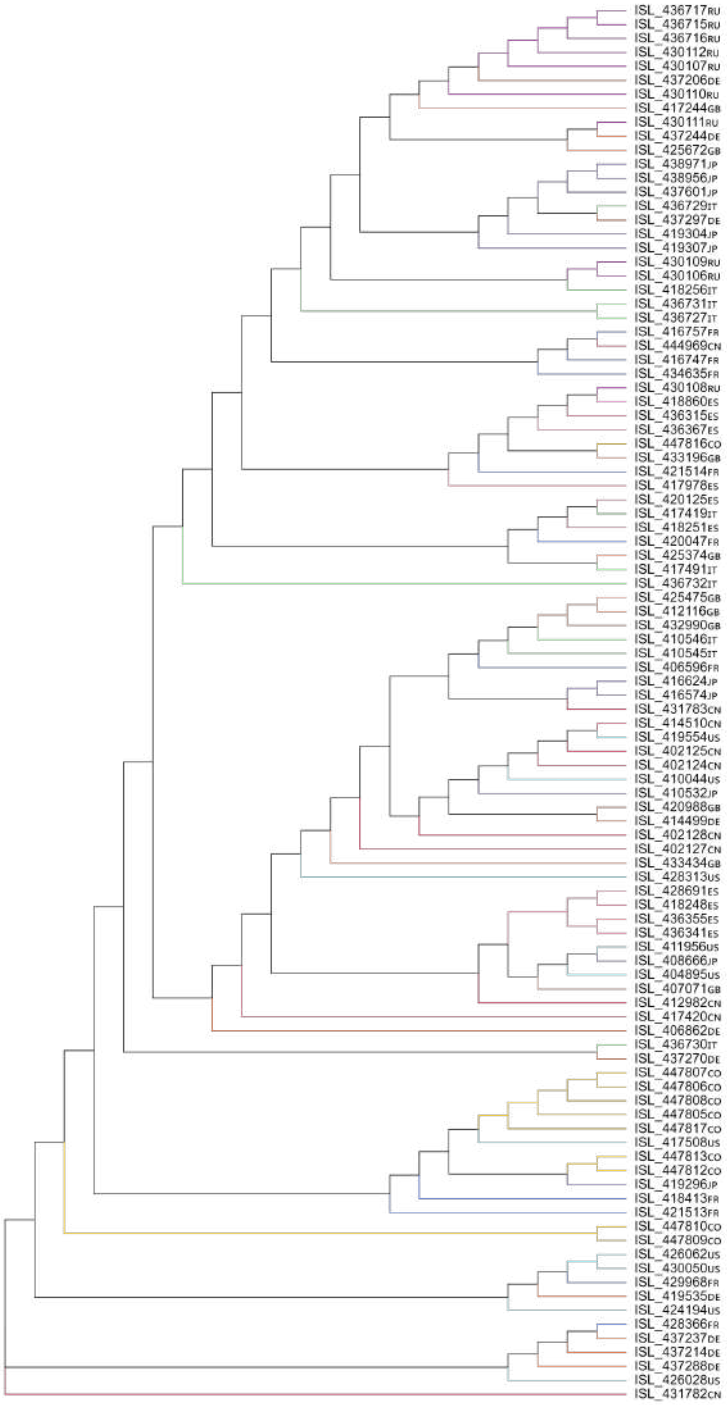
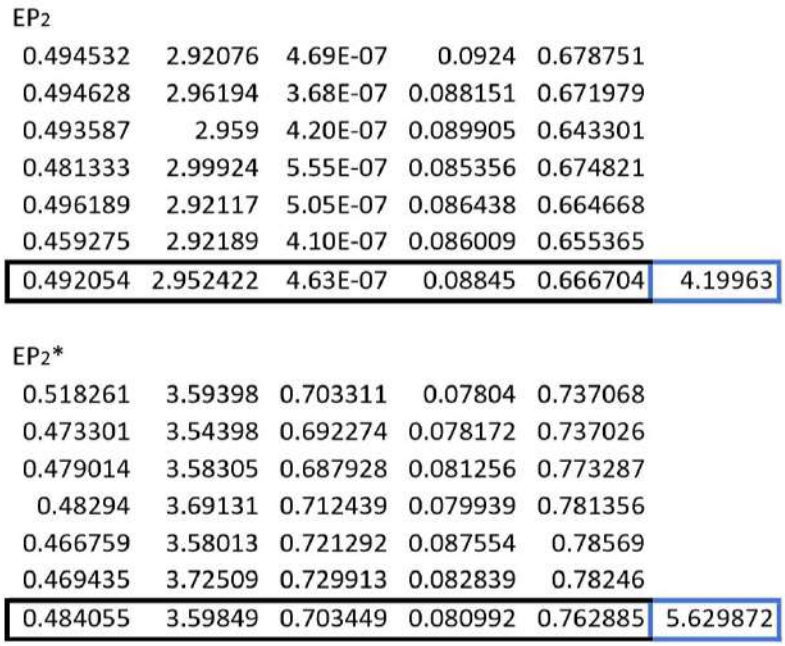


Figura 4.2: Albero prodotto dall'allineamento multiplo.



EP2

0.494532	2.92076	4.69E-07	0.0924	0.678751	
0.494628	2.96194	3.68E-07	0.088151	0.671979	
0.493587	2.959	4.20E-07	0.089905	0.643301	
0.481333	2.99924	5.55E-07	0.085356	0.674821	
0.496189	2.92117	5.05E-07	0.086438	0.664668	
0.459275	2.92189	4.10E-07	0.086009	0.655365	
0.492054	2.952422	4.63E-07	0.08845	0.666704	4.19963

EP2*

0.518261	3.59398	0.703311	0.07804	0.737068	
0.473301	3.54398	0.692274	0.078172	0.737026	
0.479014	3.58305	0.687928	0.081256	0.773287	
0.48294	3.69131	0.712439	0.079939	0.781356	
0.466759	3.58013	0.721292	0.087554	0.78569	
0.469435	3.72509	0.729913	0.082839	0.78246	
0.484055	3.59849	0.703449	0.080992	0.762885	5.629872

Figura 4.3: Sia per EP_2, sia per EP_2^*, sono state effettuate cinque computazioni, i risultati delle quali sono in ogni riga. Ogni colonna rappresenta una fase della computazione. L'ultima riga raccoglie il tempo medio di ciascuna fase, mentre il valore in basso a destra è la somma dei tempi medi.

In figura 4.3 sono mostrati i tempi di ciascuna fase.

2. La matrice prodotta da `ep_sim` è di similarità, per cui prima di passarla a `neighbor` è necessario trasformarla. Tale operazione è stata realizzata con un semplice script Python che sostituisce ciascun elemento el della matrice con
$$1 - \frac{el}{M}$$
dove M è l'elemento di valore massimo: massima similarità significa minima distanza, pari a 0. Lo stesso script, generando la matrice di distanze, si preoccupa di farlo nel formato accettato da `neighbor`: specificando il numero di sequenze e il nome di ciascuna di esse, come descritto in [48];

3. `neighbor`;

4. Iroki.

4.3 Confronto tra gli alberi

Uno scopo di questo lavoro è quello di individuare analogie e differenze tra i metodi di confronto tra le sequenze, oltre a quelle evidenti già apprezzate. Per esempio, il tempo di computazione degli allineamenti è sensibilmente

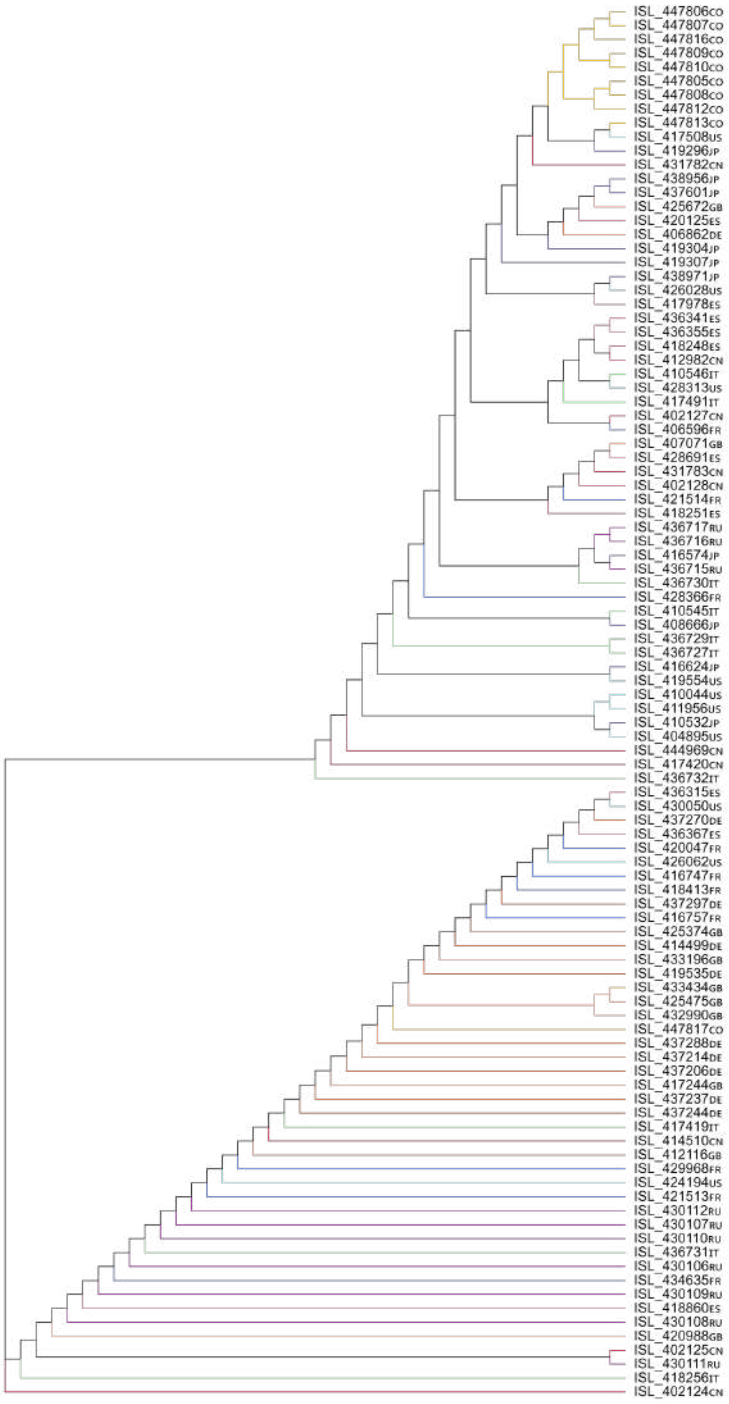


Figura 4.4: Albero prodotto da EP_2.

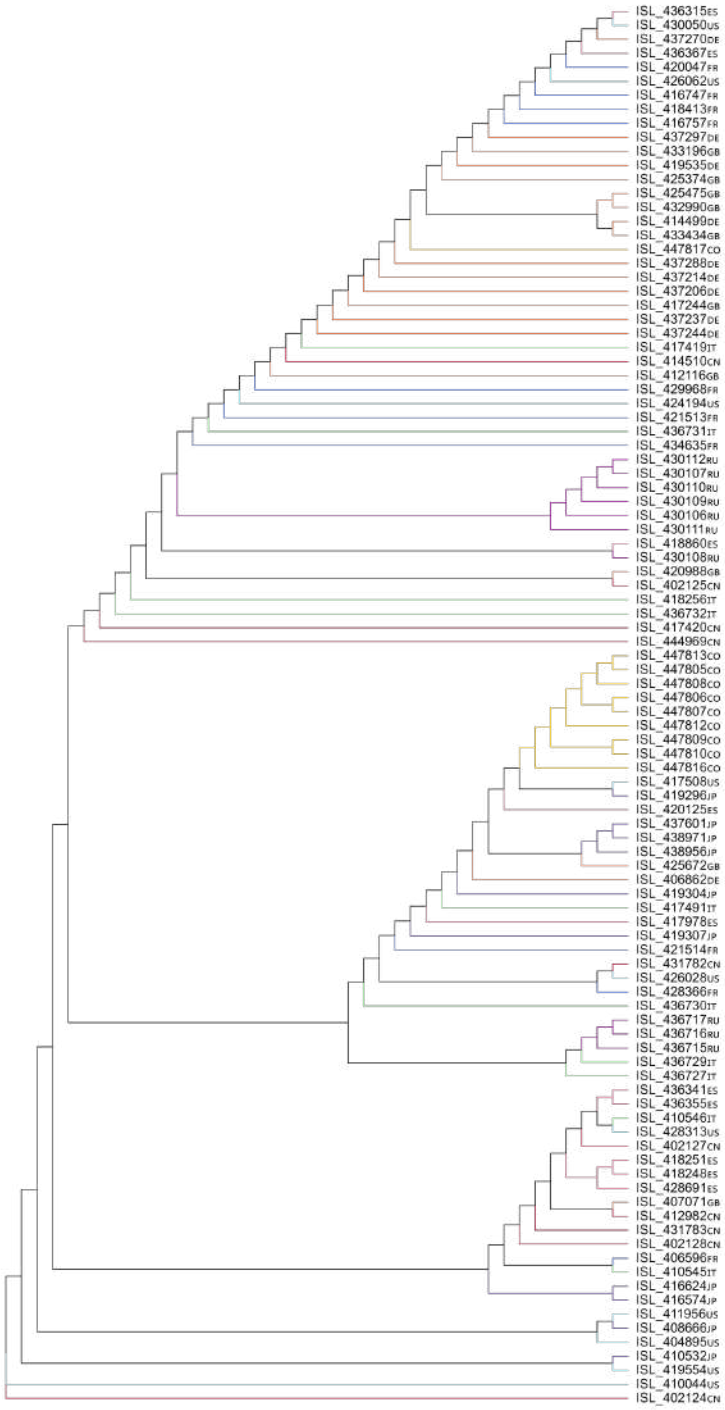


Figura 4.5: Albero prodotto da EP_2^*.

```
Tree distance program, version 3.698

Symmetric differences between all pairs of trees in tree file:
```

	1	2	3	4
1	0	84	188	172
2	84	0	188	176
3	188	188	0	134
4	172	176	134	0

Figura 4.6: L'output di `treedist` con la distanza reciproca degli alberi ottenuti da, in ordine crescente: l'allineamento a coppie, l'allineamento multiplo, `ep_sim` con `D2` e `ep_sim` con `D2*`.

superiore rispetto a quello dei metodi alignment-free, il che rende questi ultimi più adatti ad analisi su campioni di taglia maggiore. É necessario verificare, però, che i risultati ottenuti con i diversi metodi siano quanto meno confrontabili.

Per trarre conclusioni significative anche su aspetti meno evidenti, abbiamo confrontato gli alberi. Osservandoli, è possibile accorgersi da subito come siamo visibilmente diversi, ma per quantificarne la differenza sono stati adottati due metodi:

- il primo considera una distanza tra gli alberi, contando le differenze nella disposizione dei rami;
- il secondo, con un approccio di tipo statistico, non si basa sull'intera topologia dell'albero. Esso genera dei clustering, richiedendo, però, dati aggiuntivi: le classi assegnate alle sequenze. I clustering ottenuti sono valutabili singolarmente ma anche confrontabili tra loro.

Si noti, prima di proseguire, che adottando sempre lo stesso metodo per la costruzione degli alberi, NJ, non si presenta il rischio che eventuali metodi alternativi introducano ulteriori differenze intrinseche.

4.3.1 Distanza

La distanza calcolata è la distanza di Robinson-Foulds, definita nella definizione 9 e implementata da `TreeDist`, descritto in 3.5.3. I risultati del confronto tra tutte le possibili coppie di alberi è riportato in figura 4.6, sotto forma di matrice.

Secondo questa metrica, gli alberi più simili tra loro sono quelli ottenuti dagli allineamenti. Questo risultato non è inaspettato, poiché il primo

passo della computazione dell'allineamento multiplo si basa su allineamenti pairwise delle sequenze.

I due alberi ottenuti a partire dai metodi alignment-free sono, invece, distanti tra loro quasi quanto lo sono da quelli ottenuti dagli allineamenti. Ciò mette in luce la sostanziale differenza tra le statistiche D_2 e D_2^*, dato che EP_2 ed EP_2^* ne sono una generalizzazione.

Il risultato meno inaspettato è quello che riguarda le differenze tra i metodi basati sull'allineamento e quelli che non lo sono, nonostante sia interessante notare come l'albero costruito da a partire da EP_2 sia equidistante sia dall'albero ottenuto con l'allineamento pairwise, sia da quello ottenuto con l'allineamento multiplo, e che tale distanza sia la maggiore. Un discorso analogo, per 4 punti di differenza, vale per l'albero ottenuto a partire da EP_2^*.

4.3.2 Clustering

Produciamo un clustering a partire da ciascun albero, in accordo con quanto spiegato in 2.5.2. Per la valutazione si calcolano l'entropia dei singoli cluster, l'entropia media e l'entropia delle classi, visualizzandole tramite degli istogrammi. Il tool `TreeCluster` produce i clustering a partire dagli alberi mentre il tool in Python disponibile presso [17], invece, include anche il calcolo e la visualizzazione delle entropie.

Tra i metodi di `TreeCluster` ne sono stati scelti due, i quali permettono di ottenere risultati soddisfacenti:

1. `max` produce i cluster in modo tale che la massima distanza tra le foglie dell'albero all'interno dello stesso cluster sia $\leq$ `t`, soglia passata come parametro;

2. `sum_branch` seleziona ciascun cluster delle sequenze in modo che la lunghezza totale della componente connessa che lo induce (calcolata come somma delle lunghezze dei rami) sia al massimo `t`.

Le classi associate alle sequenze sono, come accennato, le nazioni di provenienza, che giustificano il volere di ottenere 10 cluster, idealmente con 10 sequenze ciascuno. In questo modo l'entropia di ogni cluster è compresa nell'intervallo $[0, \log_2 10]$ e permette di misurare la qualità dei clustering, ma allo stesso tempo verificare se i risultati giustifichino la scelta effettuata. Indirettamente, infatti, scegliendo tali classi si sta ipotizzando ci siano similarità maggiori tra sequenze di una stessa nazione piuttosto che sequenze di nazioni diverse. Se tali similarità esistessero e rispettassero quanto ipotizzato, dovrebbero essere catturate dagli alberi, e di conseguenza dai clustering prodotti. Calcolando l'entropia delle classi è possibile valutare anche la bontà di questa ipotesi.

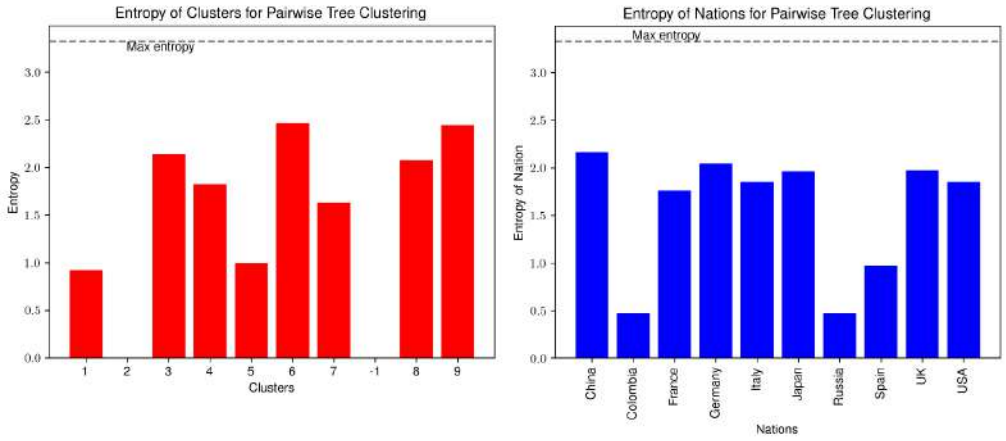


Figura 4.7: Entropia rispettivamente dei cluster prodotti da `TreeCluster` e delle classi associate, a partire dall'albero filogenetico ottenuto dall'allineamento pairwise.

Infine, data la diversità degli alberi, al parametro `t` sono assegnati valori diversi a seconda degli alberi considerati, per riuscire a ottenere sempre 10 cluster.

Allineamento

Per gli alberi filogenetici ottenuti dall'allineamento pairwise e dall'allineamento multiplo, il valore della soglia `t` per il metodo `max` di `TreeCluster` è di 0.000412 poiché tale valore permette di ottenere 10 cluster con ragionevoli valori di entropia per entrambi gli alberi. Gli istogrammi relativi alle entropie dei cluster ottenuti in questo modo sono in figura 4.7 e 4.8 rispettivamente per gli alberi filogenetici ottenuti dall'allineamento pairwise e dall'allineamento multiplo.

Allineamento pairwise Si può notare dai valori di entropia come nell'albero prodotto da allineamento pairwise siano presenti due cluster a entropia nulla: il cluster di indice 2, infatti, contiene quattro sequenze provenienti esclusivamente dalla Spagna mentre il cluster di indice -1 è un singleton.

Considerando l'entropia delle nazioni, Colombia, Russia e Spagna presentano bassi valori di entropia, per tutte minori di 1: nel caso della Colombia, osservando l'albero ottenuto, si può affermare che tutte le sequenze sono prossime tra loro, a meno di una, e derivano quindi da un unico antenato comune molto vicino. Nel caso di Russia e Spagna, invece, le sequenze derivano da antenati comuni poco distanti ma condivisi con altre nazioni. Il valore di entropia media ottenuto è di circa 1.95, suggerisce che il clustering non sia di alta qualità.

Allineamento multiplo Per il clustering relativo all'allineamento multiplo i risultati sono analoghi a quelli ottenuti con l'allineamento pairwise: il

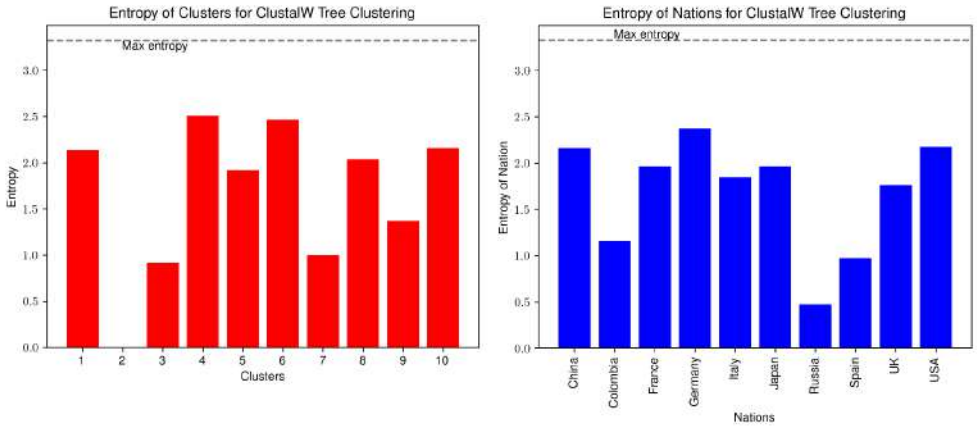


Figura 4.8: Entropia rispettivamente dei cluster prodotti da `TreeCluster` e delle classi associate, a partire dall'albero filogenetico ottenuto dall'allineamento multiplo.

cluster di indice 2 contiene solo sequenze spagnole, perciò ha entropia nulla, ed è l'unico. Le classi a minor entropia sono Colombia, Russia e Spagna. Contrariamente all'analisi precedente, però, la Colombia non è la classe a minor entropia, e ciò è probabilmente dovuto al fatto che essa presenta sequenze vicine ma con un antenato comune a maggior distanza. D'altra parte, Russia e Spagna formano gruppi di sequenze più vicine e con un antenato comune a minor distanza, giustificando la diminuzione di entropia. Il valore di entropia media in questo caso è di circa 2.06, perciò questo clustering è mediamente peggiore rispetto al precedente.

I valori di entropia ottenuti testimoniano come non sia presente una soddisfacente ripartizione delle classi. Allo stesso tempo non si può affermare che i cluster non possiedano alcun significato. Ciò è dovuto alla varietà dei risultati: Colombia e Russia sono classi che ben si prestano a questa analisi, mentre Cina e Germania, due delle classi a maggior entropia, risultano più disperse.

Alignment-free

Per gli alberi filogenetici ottenuti tramite metodi alignment-free, il valore della soglia `t` per il metodo `max` di `TreeCluster` cambia a seconda dell'albero considerato: non è possibile ottenere 10 cluster da entrambi gli alberi con un valore comune. Per l'albero ottenuto da EP_2, quindi, `t` è pari a 0.0806 mentre per quello ottenuto da EP_2^* è 0.1138. Gli istogrammi delle entropie sono in figura 4.9 e 4.10.

EP_2 Il clustering ottenuto da EP_2 presenta 7 cluster singleton, tutti contenenti una sequenza della Colombia, e solo 3 cluster contenenti un numero cospicuo di sequenze. Questo sbilanciamento, evidente anche dall'istogramma dell'entropia delle classi, non permette solide considerazioni. A giudicare

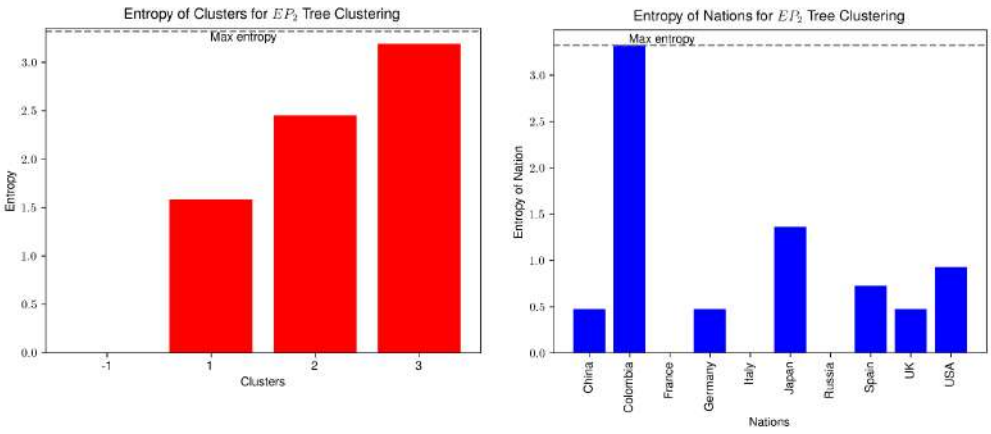


Figura 4.9: Entropia rispettivamente dei cluster prodotti da `TreeCluster` con metodo `max` e delle classi associate, a partire dall'albero filogenetico ottenuto dal confronto basato su EP_2.

dall'albero in figura 4.4 e dai clustering precedenti, è ragionevole aspettarsi che le sequenze della Colombia siano raggruppate più efficacemente di altre, di conseguenza il risultato appena ottenuto risulta eccezionale. L'entropia media, infatti, è di circa 2.83, un valore peggiore di quelli ottenuti con i clustering precedentemente considerati.

EP_2^* Il clustering ottenuto da EP_2^*, d'altra parte, sembra più ragionevole: solo tre cluster possiedono entropia nulla, e l'entropia media scende, non sorprendentemente, al valore di circa 2.31. Da questo punto di vista, i clustering ottenuti da metodi alignment-free sono peggiori di quelli ottenuti a partire dagli allineamenti. Guardando l'istogramma delle classi, però, è possibile anche confermare come le sequenze della classe Russia siano raggruppate in modo più compatto rispetto ad altre. Le sequenze della classe Spagna, tuttavia, sono più disperse, mentre quelle della classe Germania sono più raggruppate, risultato opposto a quello ottenuto in precedenza. Le sequenze della classe Colombia, anche in questo caso, non sono raggruppate come atteso.

`sum_branch`

La stessa analisi condotta con il metodo `max` può essere condotta con il metodo `sum_branch`. In questo modo, i risultati che si ottengono sugli alberi prodotti dagli allineamenti non sono significativamente diversi (si veda l'appendice D), ma per i metodi alignment-free vale la pena indagare, visto anche lo scarso successo ottenuto con il precedente metodo. I valori della soglia `t` ora sono di 0.4 per l'albero prodotto con EP_2, e di 0.65 per l'albero prodotto con EP_2^*.

Gli istogrammi sono in figura 4.11 e 4.12. Si nota subito come essi siano, per certi aspetti, più in linea con quelli ottenuti dagli alberi prodot-

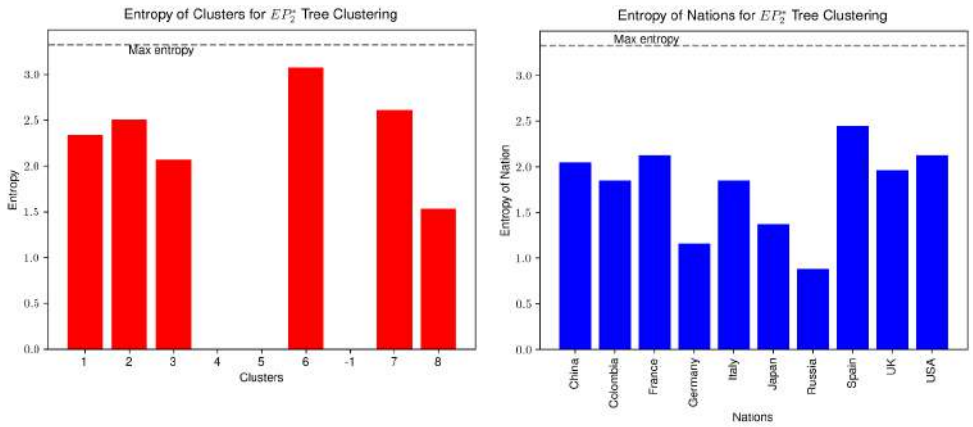


Figura 4.10: Entropia rispettivamente dei cluster prodotti da `TreeCluster` con metodo `max` e delle classi associate, a partire dall'albero filogenetico ottenuto dal confronto basato su EP_2^*.

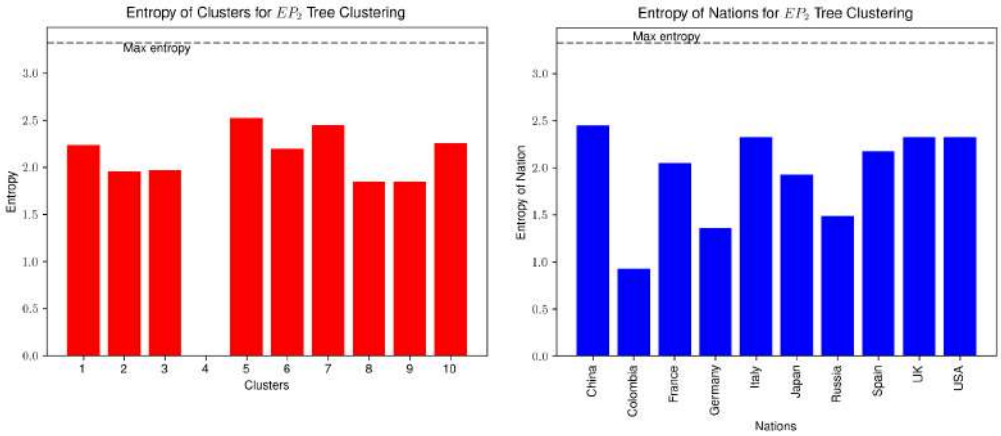


Figura 4.11: Entropia rispettivamente dei cluster prodotti da `TreeCluster` con metodo `sum_branch` e delle classi associate, a partire dall'albero filogenetico ottenuto dal confronto basato su EP_2.

ti dagli allineamenti. La loro entropia media, infatti, è migliorata: è di, rispettivamente, 1.96 e 1.88.

EP_2 Il clustering per EP_2 presenta un cluster a entropia nulla, perché contenente solo sequenze della classe Colombia, in accordo con quanto già osservato. Vale la pena notare come anche la Germania emerga rispetto alle altre nazioni, mentre Cina, Francia, Giappone, UK e USA rimangono a entropia relativamente alta. Questi risultati non sono attesi se comparati alla grossolana osservazione dell'albero, il che permette di affermare che il clustering è un buon metodo di confronto.

EP_2^* Il cluster a entropia nulla è un singleton. Alle nazioni significative si aggiunge il Giappone, che nei precedenti istogrammi ha sempre avuto un valore di entropia più simile a quello di Italia e Cina. Osservando l'albero,

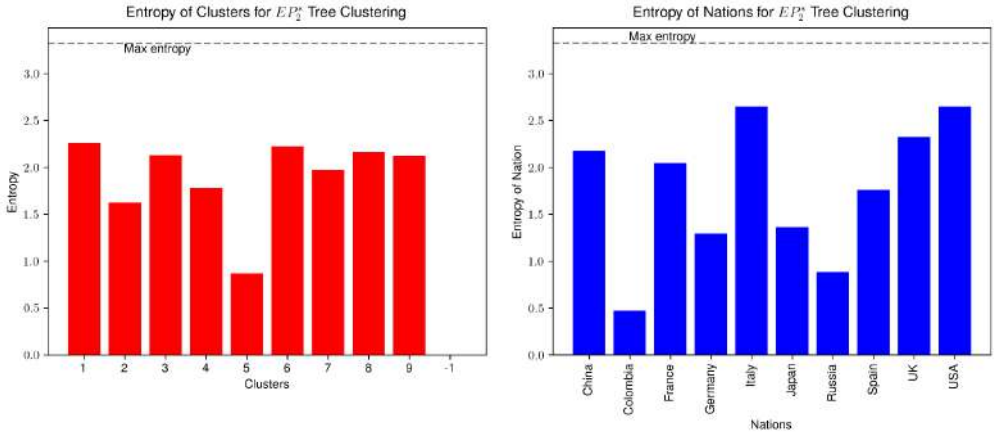


Figura 4.12: Entropia rispettivamente dei cluster prodotti da `TreeCluster` con metodo `sum_branch` e delle classi associate, a partire dall'albero filogenetico ottenuto dal confronto basato su EP_2^*.

ciò trova riscontro col fatto che le sequenze giapponesi, quando presenti, sono più spesso simili rispetto ad altre. Ciò, non essendo comunque evidente, rappresenta un risultato abbastanza inatteso, specie considerando gli alberi precedenti. Un discorso analogo si può fare per la Germania.

4.4 Considerazioni complessive

Considerando l'intera analisi i metodi concordano su alcuni risultati.

Tutti i metodi, nonostante i diversi approcci, concordano sul fatto che le sequenze della Colombia siano molto simili tra loro, e ciò è visibile già a partire dagli alberi. Per questo si può affermare, con una certa dose di sicurezza, che la mutazione che origina le sequenze colombiane è esclusiva di tali sequenze. Di conseguenza potrebbe essere una mutazione più giovane, isolata grazie alle misure di contenimento adottate dagli stati considerati. Questo non deve trarre in inganno: considerando sequenze di stati vicini alla Colombia, come Brasile o Venezuela i risultati potrebbero variare molto.

Le sequenze più sparse, secondo tutti i metodi, sono quelle prelevate da Cina, Francia, Italia, UK e USA. Il risultato è atteso per quanto riguarda le sequenze cinesi, avendo prelevato le prime disponibili. Il virus ha avuto origine proprio in Cina, quindi è naturale aspettarsi che gran parte delle altre sequenze ne condividano le mutazioni. In realtà, questa considerazione è supportata dalla conoscenza sull'origine del virus: guardando solo i dati di questa analisi nulla, se non la data di campionamento, vieterebbe di pensare all'Italia come nazione in cui il virus ha avuto origine.

Per analisi più sofisticate, si veda il sito Nextstrain, che presenta una visione più completa e costantemente aggiornata della situazione. In particolare, in [18] si mostra come non ci siano sottoalberi caratteristici tra tutte

le sequenze disponibili, permettendo anche un confronto di tipo filodinamico con il virus dell'influenza.

Tornando all'analisi, si noti come alcuni risultati siano contraddittori, perché ottenuti da alcuni metodi e subito smentiti da altri. I metodi basati sugli allineamenti dimostrano di giungere a conclusioni simili, il che suggerisce di preferire il metodo più trattabile computazionalmente, a parità di risultato. I metodi alignment-free, invece, sono in disaccordo tra loro. Ciò è probabilmente dovuto al fatto che EP_2^* rappresenta un miglioramento di EP_2, mentre i metodi basati sugli allineamenti non possiedono questa relazione. L'unico fatto, in questa analisi, che testimonia come EP_2^* produca effettivamente risultati migliori di EP_2, è quello ottenuto con il clustering con metodo `max`: EP_2 produce un clustering di bassa qualità, EP_2^*, invece, ha una qualità più simile a quella degli altri clustering. Ciò potrebbe non essere sufficiente, per cui, come approfondimento a questo lavoro, potrebbe essere interessante indagare maggiormente sulla differenza tra i due metodi concentrandosi sul loro confronto e sulla validità dei risultati prodotti.

In conclusione, a giudicare dai risultati ottenuti, gli stati non si sono rivelati classi ragionevoli, per le quali pretendere clustering di qualità. Le sequenze, infatti, seppur campionate in luoghi del pianeta distanti tra loro, sono molto simili: come già osservato durante la computazione di ClustalW2, esse condividono più del 90% dei simboli. Se le mutazioni fossero avvenute indipendentemente in ogni stato, forse avremmo ottenuto dei cluster di maggiore qualità: questo scenario suppone che non ci siano stati spostamenti tra gli stati durante il tempo richiesto a una sequenza per mutare in modo significativo.

La scelta degli stati come classi, d'altra parte, è una delle poche scelte intuitive possibili. Se gli alberi e i clustering hanno prodotto ciò che abbiamo riportato, significa che ciò che accomuna maggiormente le sequenze è altro, probabilmente ricostruibile (e validabile) solo tracciando una dinamica dei contagi tra gli stati scelti per l'analisi. Da [19], tuttavia, è possibile osservare come non esista alcuna etichetta che permetta di distinguere in modo netto dei gruppi di sequenze.

Infine, si sono osservati anche casi di sequenze sensibilmente diverse da quelle considerate in questo lavoro, ma si tratta di casi isolati [36].

Capitolo 5

Conclusioni

Le diverse sequenze di SARS-CoV-2 considerate sono simili per il 90% dei caratteri. Questo significa che in ogni prelievo, seppur distante nel tempo e nello spazio, il genoma del virus è all'incirca lo stesso. In altre parole, si può affermare che gli esseri umani si sono trasmessi il virus più velocemente di quanto il virus stesso sia riuscito a mutare in modo significativo [20].

Con una struttura così simile, gli algoritmi sono messi a dura prova nell'individuare le differenze tra le sequenze e metterle in risalto. Gli allineamenti si propongono di confrontare le sequenze in modo più preciso, sebbene l'ottimo non sia garantito. Oltre a questo, anche il tempo di computazione risulta spesso impraticabile. Dall'altra parte, i metodi alignment-free potrebbero essere particolarmente adatti proprio per questo tipo di analisi, grazie al loro tempo di computazione irrisorio, se confrontato con gli allineamenti, e all'idea sulla quale sono basati. Bisogna, però, menzionarne uno svantaggio: la grande quantità di parametri da considerare che possono incidere significativamente sui risultati. Proprio a causa di questo motivo si è rivelato indispensabile adottare diversi metodi: in caso di incertezza si possono valutare e confrontare i risultati prodotti, individuando quelli più solidi rispetto alle anomalie.

5.1 Possibili approfondimenti

In futuro si avrà modo di tracciare una dinamica più chiara dell'evoluzione del virus, permettendo una valutazione più accurata, coprendo anche luoghi del mondo per i quali attualmente non sono disponibili sufficienti dati. Sarebbe inoltre interessante approfondire l'analisi non solo sulla base della provenienza geografica delle sequenze, ma anche su altri dati correlati a esse, come la data di campionamento o i dati clinici del paziente. Alcuni studi pubblicati che già realizzano questi propositi sono menzionati e citati nelle precedenti sezioni.

In generale, le analisi condotte produrrebbero risultati di maggiore validità statistica se ripetute considerando un campione più rappresentativo e con più sequenze. Ciò necessiterebbe inevitabilmente di maggiore potenza di calcolo rispetto a quella utilizzata in questa sede, almeno per gli allineamenti, che sono le fasi del lavoro più onerose.

Indipendentemente da ciò, si può ampliare l'analisi intervenendo su diversi passaggi del lavoro.

Si possono utilizzare altri algoritmi per l'allineamento, per i confronti alignment-free e anche per il confronto tra alberi, adottando misure e varianti proposte nelle varie sezioni e appendici di questo elaborato.

Si può studiare anche come variano i risultati in funzione dei parametri scelti nei tool utilizzati, di seguito un paio di esempi:

- in `ep_sim` sono presenti molti parametri che sono stati lasciati ai valori di default, come il modello statistico, la varianza dei pesi gaussiani e la scelta stessa dei pesi;

- negli algoritmi di allineamento sono usate matrici di score, gap opening penalty e gap extension penalty, che possono essere modificati in accordo con informazioni più specifiche su SARS-CoV-2.

Si può indagare sul metodo di costruzione degli alberi, usando anche UPGMA, implementato negli stessi tool adottati in questo lavoro, confrontandone poi i risultati con quelli ottenuti con NJ.

Per quanto riguarda il clustering, si possono adottare ulteriori strategie, non solo a partire dagli alberi, ma anche dalla matrice o direttamente dalle sequenze.

Infine, l'entropia non è l'unico metodo di valutazione di clustering possibile, ne esistono anche altri, come la valutazione non supervisionata o la statistica di Hopkins (si veda l'appendice B.5).

Tutto ciò giustifica la scelta di entrare nei dettagli di ogni aspetto del lavoro svolto, facilitandone l'estensione su più punti, e anche di inserire un'appendice alla quale fare riferimento per lo scopo.

Appendice A

Informazioni sulle sequenze

In questo capitolo sono presenti, in formato tabellare, alcuni dati riguardanti le sequenze impiegate nell'analisi. La tabella A.1 contiene i dati, mentre la tabella A.2 mappa le nazioni delle sequenze con la sigla e il colore associati, per poterli distinguere negli alberi filogenetici. Tutte le sequenze e i dati completi sono disponibili presso Gisaid [13].

ID	**Luogo**	**Data**
ISL_402124	China / Hubei / Wuhan	2019-12-30
ISL_402125	China	2019-12-31
ISL_402127	China / Hubei / Wuhan	2019-12-30
ISL_402128	China / Hubei / Wuhan	2019-12-30
ISL_404895	USA / Washington / Snohomish County	2020-01-19
ISL_406596	France / Ile-de-France / Paris	2020-01-23
ISL_406862	Germany / Bavaria / Munich	2020-01-28
ISL_407071	United Kingdom / England	2020-01-29
ISL_408666	Japan / Tokyo	2020-01-31
ISL_410044	USA / California	2020-01-27
ISL_410532	Japan / Osaka	2020-01-23
ISL_410545	Italy / Rome	2020-01-29
ISL_410546	Italy / Rome	2020-01-31
ISL_411956	USA / Texas	2020-02-11
ISL_412116	United Kingdom / England	2020-02-09
ISL_412982	China / Hubei / Wuhan	2020-02-07
ISL_414499	Germany / North Rhine Westphalia / Heinsberg District	2020-02-26
ISL_414510	China / Shanghai	2020-02-02
ISL_416574	Japan / unknown	2020-02-15
ISL_416624	Japan / unknown	2020-02-17
ISL_416747	France / ARA	2020-03-04
ISL_416757	France / ARA	2020-03-07

ID	Luogo	Data
ISL_417244	United Kingdom / England	2020-03-03
ISL_417419	Italy / Friuli-Venezia Giulia	2020-03-01
ISL_417420	China / NanChang	2020-03-23
ISL_417491	Italy / Marche	2020-03-03
ISL_417508	USA / Wisconsin	2020-03-16
ISL_417978	Spain / Madrid	2020-03-09
ISL_418248	Spain / Castilla y Leon	2020-03-01
ISL_418251	Spain / Madrid	2020-02-25
ISL_418256	Italy / Abruzzo	2020-03-14
ISL_418413	France / ARA / Macon	2020-03-15
ISL_418860	Spain / Barcelona	2020-03-15
ISL_419296	Japan / Kochi	2020-03-08
ISL_419304	Japan / Saitama	2020-03-17
ISL_419307	Japan / Saitama	2020-03-20
ISL_419535	Germany / Duesseldorf	2020-03-13
ISL_419554	USA / California	2020-02-26
ISL_420047	France / Ile de France / Gennevilliers	2020-03-17
ISL_420125	Spain / Comunitat Valenciana / Valencia	2020-03-11
ISL_420988	United Kingdom / Wales	2020-03-23
ISL_421513	France / Ile de France / Orsay	2020-03-23
ISL_421514	France / Centre - Val de Loire / Abondant	2020-03-20
ISL_424194	USA / Washington	2020-03-20
ISL_425374	United Kingdom / England	2020-03-19
ISL_425475	United Kingdom / England	2020-03-14
ISL_425672	United Kingdom / Scotland	2020-03-11
ISL_426028	USA / New York	2020-03-04
ISL_426062	USA / Idaho	2020-03-24
ISL_428313	USA / Wisconsin / Milwaukee	2020-03-25
ISL_428366	France / Hauts De France / Chateau-thierry	2020-03-30
ISL_428691	Spain / Madrid	2020-03-21
ISL_429968	France / Hauts de France / Compiegne	2020-02-21
ISL_430050	USA / Utah	2020-03-31
ISL_430106	Russia / St.Petersburg	2020-04-15
ISL_430107	Russia / St.Petersburg	2020-04-15
ISL_430108	Russia / St.Petersburg	2020-04-15
ISL_430109	Russia / St.Petersburg	2020-04-15
ISL_430110	Russia / St.Petersburg	2020-04-15
ISL_430111	Russia / St.Petersburg	2020-04-15
ISL_430112	Russia / Buryat Republic / Ulan-Ude	2020-03-25
ISL_431782	China / Fujian	2020-03-22
ISL_431783	China / Fujian	2020-03-19

ID	Luogo	Data
ISL_432990	United Kingdom / England	2020-04-01
ISL_433196	United Kingdom / Scotland	2020-04-19
ISL_433434	United Kingdom / Scotland	2020-04-09
ISL_434635	France / Occitanie	2020-04-11
ISL_436315	Spain / Comunitat_Valenciana / Valencia	2020-03-27
ISL_436341	Spain / Comunitat_Valenciana / Valencia	2020-03-29
ISL_436355	Spain / Comunitat_Valenciana / Valencia	2020-03-26
ISL_436367	Spain / Comunitat_Valenciana / Valencia	2020-03-25
ISL_436715	Russia / Moscow	2020-04-14
ISL_436716	Russia / Moscow	2020-04-14
ISL_436717	Russia / Moscow	2020-04-14
ISL_436727	Italy / Abruzzo	2020-04-27
ISL_436729	Italy / Abruzzo	2020-04-27
ISL_436730	Italy / Abruzzo	2020-04-27
ISL_436731	Italy / Abruzzo	2020-04-26
ISL_436732	Italy / Abruzzo	2020-04-27
ISL_437206	Germany / Bavaria / Munich	2020-03-18
ISL_437214	Germany / Bavaria / Munich	2020-04-07
ISL_437237	Germany / Bavaria / Munich	2020-03-23
ISL_437244	Germany / Bavaria / Munich	2020-03-08
ISL_437270	Germany / Bavaria / Munich	2020-03-29
ISL_437288	Germany / Bavaria / Munich	2020-04-03
ISL_437297	Germany / Bavaria / Munich	2020-04-13
ISL_437601	Japan / Kanto	2020-03-31
ISL_438956	Japan/ Kanto	2020-04-08
ISL_438971	Japan/ Kanto	2020-04-24
ISL_444969	China / Guangdong / Guangzhou	2020-04-16
ISL_447805	Colombia / Boyaca / Villavicencio	2020-04-04
ISL_447806	Colombia / Huila / Villavicencio	2020-04-04
ISL_447807	Colombia / Huila / Villavicencio	2020-04-04
ISL_447808	Colombia / Tolima / Togui	2020-04-04
ISL_447809	Colombia / Risaralda / Neiva	2020-04-04
ISL_447810	Colombia / Risaralda / Neiva	2020-04-05
ISL_447812	Colombia / Cundinamarca / Pereira	2020-04-05
ISL_447813	Colombia / Cundinamarca / Neiva	2020-04-05
ISL_447816	Colombia / Cordoba / Soacha	2020-04-07
ISL_447817	Colombia / Cordoba / Yumbo	2020-04-09

Tabella A.1: Le voci della tabella sono: identificatore, luogo e data del prelievo del campione poi sequenziato. Le sequenze sono inserite in ordine crescente rispetto l'identificatore.

Nazione	**Codice**	**Colore**
Cina	CN	Rosso
Colombia	CO	Giallo
Francia	FR	Blu
Germania	DE	Arancione
Italia	IT	Verde
Giappone	JP	Violetto
Russia	RU	Magenta
Spagna	ES	Rosa
Regno Unito	GB	Salmone
Stati Uniti	US	Azzurro

Tabella A.2: Legenda dei colori utilizzati per distinguere le nazioni associate alle sequenze, lungo i rami degli alberi filogenetici.

Appendice B

Approfondimenti

Di seguito sono riportati alcuni approfondimenti di argomenti trattati nel corso di questo lavoro, ma che non sono direttamente coinvolti nell'analisi condotta. Possono, di conseguenza, costituire il punto di partenza per un ampliamento dell'analisi, oppure per un completamento del corredo teorico già presentato.

B.1 Statistica N_2

Proponiamo un'altra statistica, chiamata N_2, che non si limita a effettuare il conto dei k-mer, ma prende in considerazione anche i *vicini*: si consideri una parola $w \in \Omega^{k>0}$, definiamo $n(w)$ l'insieme di parole che sono in qualche modo collegate a w, per esempio

$$n(w) = \{w' \in \Omega^k : d_{\mathcal{H}}(w, w') = 1\}$$

dove $d_{\mathcal{H}}$ è la distanza di Hamming. Il conto pesato per $n(w)$ è definito come

$$N_{n(w)} = \sum_{w' \in n(w)} a_{w'} N_{w'}$$

dove $a_{w'}$ è un peso assegnato alla parola w'. Standardizzando il conto si ottiene

$$\tilde{N}_w = \frac{N_{n(w)} - \mathbb{E}[N_{n(w)}]}{\sqrt{Var[N_{n(w)}]}},$$

normalizzando

$$\hat{N}_w = \frac{\tilde{N}_w}{\|\tilde{N}_w\|},$$

infine si giunge alla definizione

Definizione 13 (N_2)**.**

$$N_2 = \sum_w \hat{N}_w^X \hat{N}_w^Y. \tag{B.1}$$

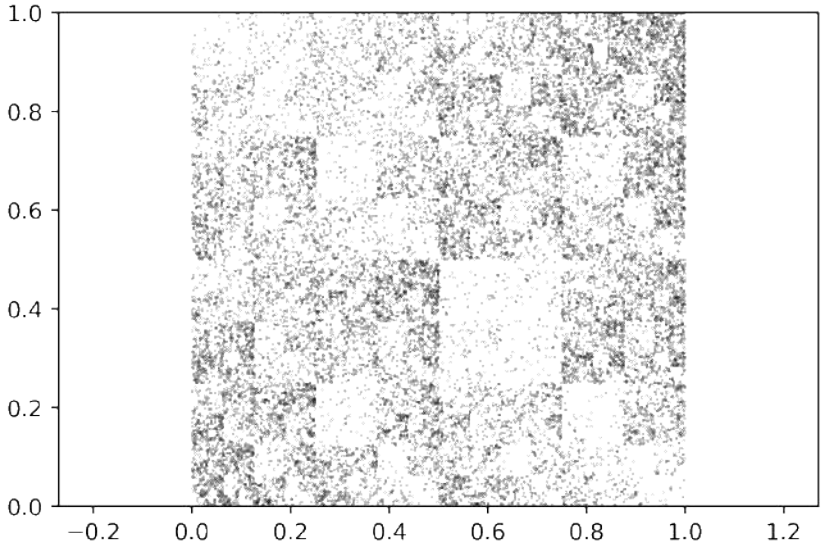


Figura B.1: CGR della sequenza in [23].

B.2 Script Python per CGR

Con l'intento di implementare la mappa CGR è stato scritto questo semplice script e testato su una sequenza di SARS-CoV-2, il cui risultato è in figura B.1. Il quadrato risulta meno denso di punti rispetto alle immagini 2.5 perché la sequenza usata ha molti meno caratteri di quelle di Synechococcus sp e Bacillus subtilis. Nonostante questo, si possono comunque apprezzare i sottoquadrati di diverse dimensioni e densità.

Il codice legge un file di testo contenente solo i caratteri della sequenza, supponendo che essi siano tutti noti: solo A, C, G, e T. Al termine della sequenza è inserito un carattere che indica la fine della lettura. Viene poi costruito un quadrato unitario e la poligonale ha come punto iniziale un punto scelto casualmente, con `uniform.rvs()` di `scipy.stats`. Il resto è l'implementazione di quanto definito in 2.5.

```
import matplotlib.pyplot as plt
from scipy.stats import uniform

def cgr(dna_translated):
    dna_x = dna_translated[0]
    dna_y = dna_translated[1]
    x = [uniform.rvs()]
    y = [uniform.rvs()]
    for i in range(len(dna_x)):
        x.append(x[i] + 0.5*(dna_x[i]-x[i]))
```

```
            y.append(y[i] + 0.5*(dna_y[i]-y[i]))
        return [x, y]

def translate_dna(dna):
    dna_x = []
    dna_y = []
    for i in range(len(dna)):
        if dna[i] == 'A':
            dna_x.append(0)
            dna_y.append(0)
        elif dna[i] == 'C':
            dna_x.append(0)
            dna_y.append(1)
        elif dna[i] == 'G':
            dna_x.append(1)
            dna_y.append(0)
        elif dna[i] == 'T':
            dna_x.append(1)
            dna_y.append(1)
    return [dna_x, dna_y]

dna = []
with open('NC_045512.2.txt') as file:
    read = file.read()
    for i in read:
        el = i.upper() # suppose there are not 'N' characters
        if el == 'A' or el == 'C' or el == 'G' or el == 'T':
            dna.append(el)
        elif el == 'N':
            print('Reading end.')
chaos_game_data = cgr(translate_dna(dna))
plt.plot(chaos_game_data[0], chaos_game_data[1],
         '.k', markersize=0.2)
plt.axis('Equal')
plt.xlim(0, 1)
plt.ylim(0, 1)
plt.savefig('NC_045512.2.png')
```

B.3 UPGMA

UPGMA (*Unweighted Pair Group Method with Arithmetic mean*) è un algoritmo di costruzione di alberi filogenetici basato sulle distanze che intercorrono tra le sequenze analizzate: è possibile generare alberi radicati, in

formato di dendrogramma. Per la successiva analisi si ricordi che in generale sono definiti cluster insiemi di una o più sequenze analizzate. Inizialmente, quindi, le singole sequenze possono essere considerate come cluster di cardinalità 1, cioè singleton. La matrice delle distanze fornita in ingresso all'algoritmo definisce quindi le distanze tra tali cluster di singole sequenze.

Il procedimento comincia con il posizionamento delle sequenze in ingresso in nodi foglia. L'algoritmo procede iterativamente associando coppie di cluster a distanza minima: dopo aver determinato quale sia tale coppia, i due cluster rappresentati nell'albero grazie all'inserimento di un nodo rappresentante l'antenato comune tra i due cluster. Al nodo antenato, sono connessi i nodi rappresentanti i due cluster tramite archi di lunghezza pari alla metà della distanza che intercorre tra essi. La coppia di cluster è ora considerata come il singolo nuovo cluster antenato comune ed è necessario aggiornare le distanze di tale nuovo cluster da tutti gli altri: la dimensione della matrice delle distanze viene dunque ridotta di un un'unità e i valori delle distanze sono aggiornati secondo quanto sotto riportato. Tale procedimento continua fino all'iterazione in cui si crea un unico cluster comprendente tutte le sequenze, il quale rappresenterà quindi la radice dell'albero.

Siano C ed G due generici cluster. La distanza tra di essi è definita come la distanza media tra tutte le possibili coppie di sequenze (X, Y) con $X \in C$ e $Y \in G$:

$$d_{C,G} = \frac{1}{\|C\| \cdot \|G\|} \sum_{X \in C} \sum_{Y \in G} d(X, Y).$$

In altre parole, se in una generica iterazione dell'algoritmo i cluster A e B sono a minima distanza e sono uniti in un unico cluster C, la distanza tra esso e ogni altro cluster G presente nella matrice delle distanze è aggiornata secondo la formula:

$$d_{C=A\cup B,G} = \frac{\|A\|}{\|A\| + \|B\|} d_{A,G} + \frac{\|B\|}{\|A\| + \|B\|} d_{B,G}$$

dove $d_{A,G}$ e $d_{B,G}$ sono note all'inizio dell'iterazione. I valori aggiornati sono quindi dati dalla media ponderata delle distanze dei cluster, pesati con la frazione di sequenze che essi contengono.

É importante sottolineare il fatto che determinando la distanza tra due cluster, essa per costruzione indichi la distanza media tra le sequenze contenute nei cluster, cioè la distanza tra le foglie dell'albero, e non tra i cluster stessi, vale a dire tra i nodi interni. Per questo motivo, creando un nodo interno all'albero relativo a un nuovo antenato comune, la distanza da calcolare dal cluster che lo compongono è da scalare con la distanza di questi ultimi cluster dai loro nodi foglia.

	a	b	c	d	e
a	0	17	21	32	23
b	17	0	30	34	21
c	21	30	0	28	39
d	31	34	28	0	43
e	23	21	39	43	0

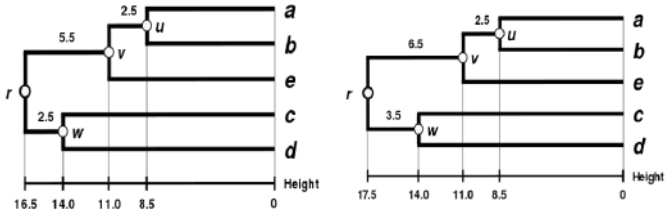


Figura B.2: Alberi risultati dalla matrice riportata a sinistra: in centro tramite UPGMA, a destra tramite WPGMA [49].

B.3.1 WPGMA

L'algoritmo WPGMA procede in modo analogo a quanto appena descritto, ma considera in modo differente l'aggiornamento dei valori di distanza: le distanze non sono pesate con il rapporto tra le cardinalità, ma si considera solamente una media aritmetica tra esse:

$$d_{C=A\cup B,G} = 1/2(d_{A,G} + d_{B,G}).$$

B.4 Clustering spettrale

Sia W una matrice di similarità tra le N sequenze di un insieme I.

Definizione 14 (Grafo di similarità)**.** Il grafo di similarità $\mathcal{G} = (V, E)$ associato alla matrice W è un grafo non diretto tale che $V = I$ e il cui peso di un ramo $(u, v) \in E$ vale $w(u, v) = W_{u,v} \geq 0$.

Un modo per ottenere un clustering $\{L_1, \ldots, L_k\}$ delle sequenze in I è quello di partire da tale grafo e rimuovere i rami per ottenere $k > 0$ componenti connesse $L_{i\in[1,k]}$ che minimizzino la funzione

$$\sum_{i=1}^{k} \sum_{u\in L_i, v\notin L_i} W_{u,v}. \tag{B.2}$$

Per $k = 2$ il problema può essere risolto efficientemente [45]. In pratica, tuttavia, esso fornisce spesso delle partizioni che non sono significative, separando vertici singoli dal resto del grafo, ottenendo in gran parte dei cluster singleton. Questo non è in linea con l'obiettivo per cui si risolve un problema di clustering, cioè quello di ottenere dei gruppi quantomeno bilanciati, la cui dimensione sia confrontabile.

Esistono molte soluzioni proposte, che prevedono ad esempio di normalizzare la funzione B.2 o di trasformare gli elementi della matrice W oppure di usare la matrice *unnormalized graph Laplacian*. Anziché soffermarci su

```
[0 0 0 0 0 1 0 0 0 0
 6 4 5 2 8 7 2 9 3 1
 0 1 1 1 1 0 0 0 1 0
 0 1 1 1 1 1 1 1 1 1
 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 1 0 0 1
 0 1 1 0 1 1 0 1 1 1
 0 0 0 0 0 0 0 1 0 1
 0 0 1 0 0 0 0 0 0 0]
```

Figura B.3: L'output dell'algoritmo consiste in un indice associato a ciascuna sequenza, che rappresenta il cluster di appartenenza. I cluster di indice `0` e `1` sono i più numerosi, gli altri sono quasi tutti singleton.

tali soluzioni, mostriamo il risultato dell'applicazione di un algoritmo di risoluzione del problema, applicato sulla matrice di similarità prodotta da `ep_sim` con EP_2. Più specificatamente, si tratta di `SpectralClustering()` di `sklearn`, la cui documentazione è in disponibile presso [21]. La comodità nell'utilizzo di questo metodo sta nel poter specificare il numero di cluster richiesto. Purtroppo, il risultato, mostrato in figura è in linea con quanto appena affermato: non si ottiene un cluster omogeneo, ma tanti singleton. Questo giustifica anche la scelta di effettuare clustering a partire dagli alberi filogenetici.

```
from sklearn.cluster import SpectralClustering
import numpy as np
inp = input()
arr = np.array([[x for x in inp.split()]])
for i in range(1, 100):
        inp = input()
        arr = np.append(arr,
          np.array([[x for x in inp.split()]]),
                                        axis=0)
clustering = SpectralClustering(n_clusters=10,
                      affinity='precomputed',
                  assign_labels="discretize",
              random_state=0).fit_predict(arr)
print(clustering)
```

B.5 Statistica di Hopkins

Quando si ha a che fare con un problema di clustering, è bene verificare che i punti dell'insieme di input ben si prestino a essere raggruppati in cluster.

Per misurare la *tendenza* al clustering si utilizza la *statistica di Hopkins*. Data una distanza d opportuna, sia I l'insieme di N sequenze in input appartenente allo spazio metrico (Ω^n, d), fissando $n \in \mathbb{N}$ per semplicità. Fissando il parametro $t \ll N$, si definiscono

- $\{\lambda_1, \ldots, \lambda_t\} \subset I$ l'insieme di sequenze prese casualmente da I;
- $\{\gamma_1, \ldots, \gamma_t\} \subset \Omega^n$ l'insieme di sequenze prese casualmente da Ω^n.

Data la distanza punto-insieme $d(p, S) = \min_{q \in S} d(p, q)$, siano, per ogni $i \in [1, t]$,

- $\beta_i = d(\lambda_i, I/\{\lambda_i\})$;
- $\tau_i = d(\gamma_i, I/\{\gamma_i\})$.

É possibile ora definire la statistica di Hopkins.

Definizione 15 (Statistica di Hopkins)**.**

$$H(I) = \frac{\sum_{i=1}^{t} \tau_i}{\sum_{i=1}^{t} \tau_i + \sum_{i=1}^{t} \beta_i}. \tag{B.3}$$

A seconda del valore che assume, tale distanza esprime quanto l'insieme in input sia simile a un insieme generato casualmente, e quindi poco incline a essere partizionato in modo significativo. Per essa, si può dimostrare che la seguente proprietà è verificata:

Proprietà 3.

$$H(I) \in [0, 1] \begin{cases} 1 \text{ cluster definiti e separati,} \\ \frac{1}{2} \text{ insieme generato casualmente,} \\ 0 \text{ dati uniformemente distribuiti.} \end{cases} \tag{B.4}$$

In termini pratici, per $H(I) > 1/2$, c'è una certa probabilità che I abbia un significato statistico.

Appendice C

Alberi

In questo capitolo sono riportati alcuni alberi ottenuti cambiando i parametri di Iroki.

C.1 Alberi circolari

Le figure C.1, C.2, C.3 e C.3 riportano alberi circolari ottenuti con gli stessi procedimenti descritti nel capitolo 4 ma utilizzando le seguenti opzioni di visualizzazione per Iroki:

- `layout shape circular`
- `branch cladogram`
- `not sorted`
- `size 40`
- `padding` minimo
- `no scale bar`
- `show leaf labels`
- `branch options` minimo
- tutti gli altri parametri lasciati di default.

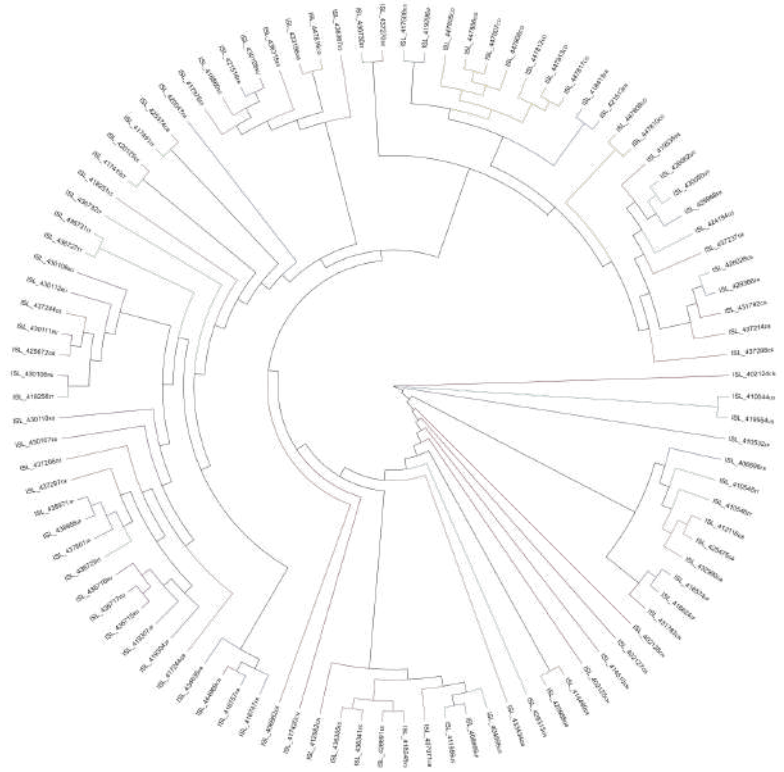


Figura C.1: Albero circolare prodotto da allineamento a coppie.

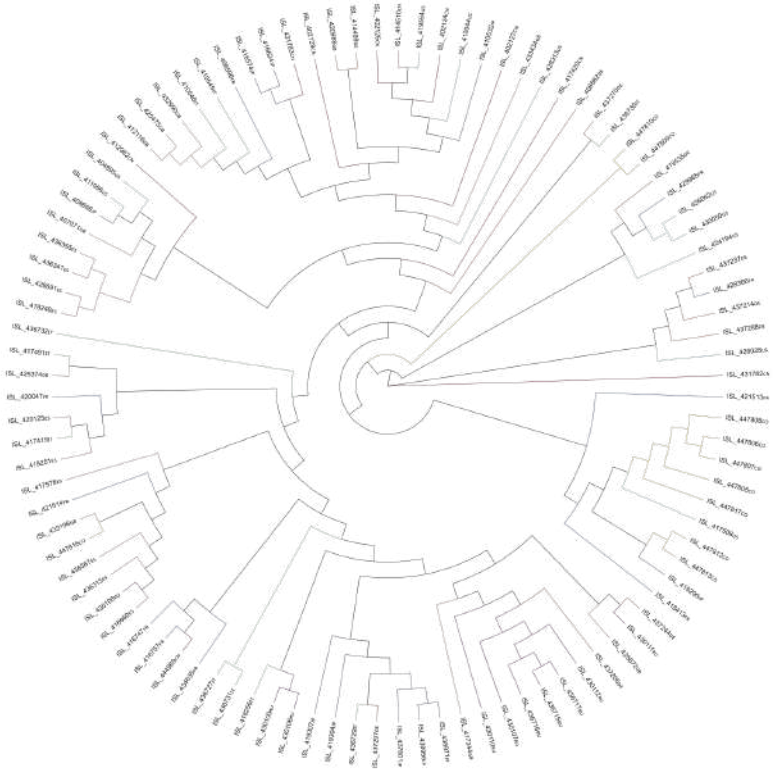


Figura C.2: Albero circolare prodotto dall'allineamento multiplo.

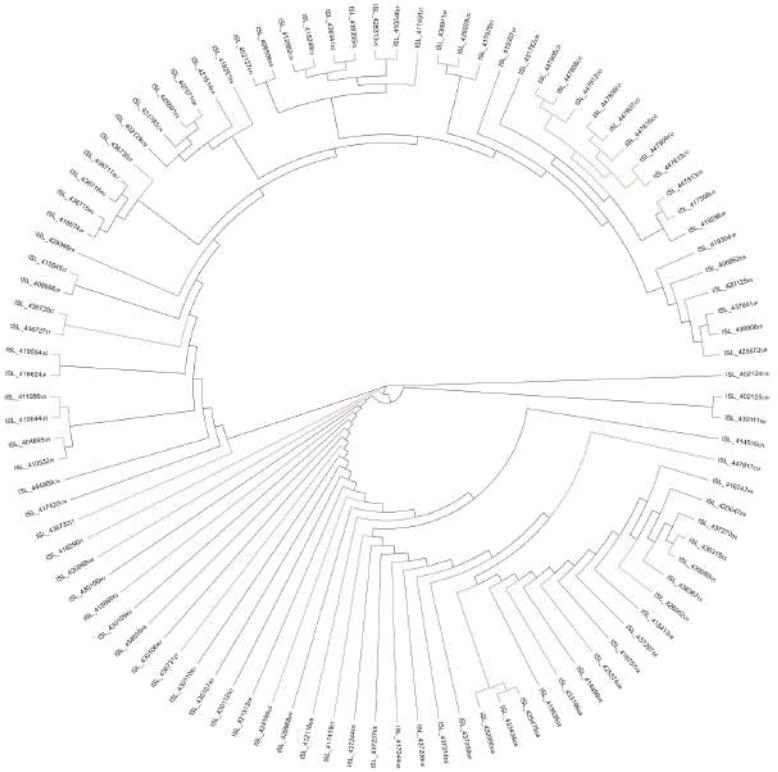


Figura C.3: Albero circolare prodotto da EP_2.

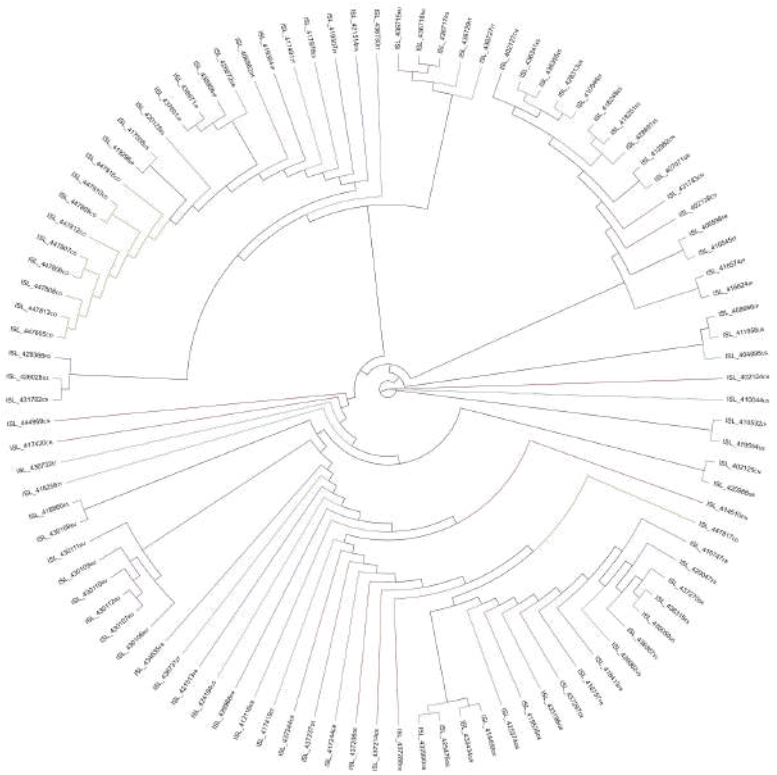


Figura C.4: Albero circolare prodotto da EP_2^*.

Appendice D

Istogrammi entropie

In questo capitolo sono presenti gli istogrammi delle entropie dei clustering prodotti a partire dagli alberi filogenetici ottenuti con l'allineamento pairwise e multiplo, con metodo `sum_branch` di `TreeCluster` e `t` pari a 0.00071. Si trovano in figura rispettivamente D.1 e D.2.

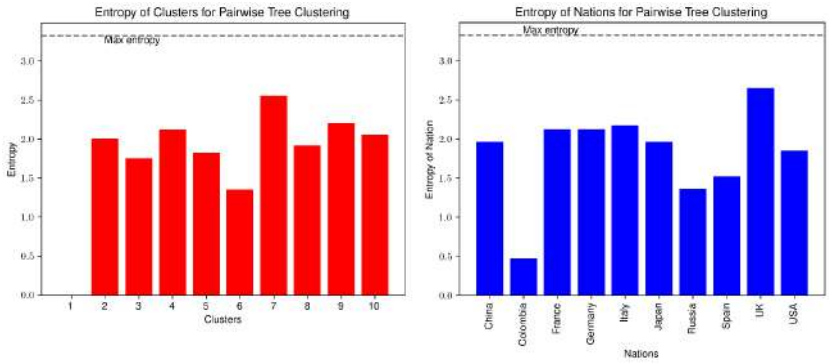


Figura D.1: Entropia rispettivamente dei cluster prodotti da `TreeDist` con metodo `sum_branch` e delle classi associate, a partire dall'albero filogenetico ottenuto dall'allineamento pairwise.

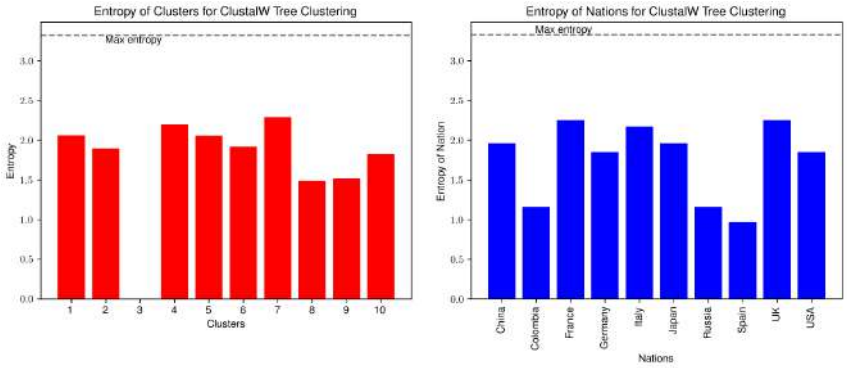


Figura D.2: Entropia rispettivamente dei cluster prodotti da `TreeDist` con metodo `sum_branch` e delle classi associate, a partire dall'albero filogenetico ottenuto dall'allineamento multiplo.

Bibliografia

[1] https://phil.cdc.gov/Details.aspx?pid=23312.

[2] http://www.g-language.org/wiki/cgr.

[3] https://towardsdatascience.com/.

[4] http://biopython.org/DIST/docs/tutorial/Tutorial.html.

[5] http://emboss.sourceforge.net/index.html.

[6] http://emboss.sourceforge.net/apps/release/6.6/emboss/apps/stretcher.html.

[7] http://emboss.sourceforge.net/apps/release/6.6/embassy/index.html.

[8] http://emboss.sourceforge.net/apps/release/6.6/embassy/phylipnew.

[9] https://www.genome.jp/tools/clustalw/clustalw_help.html.

[10] http://power.nhri.org.tw/power/OptionClustal.htm.

[11] https://www.seqan.de/.

[12] https://github.com/niemasd/TreeCluster.

[13] https://www.gisaid.org/.

[14] https://github.com/tommaso-green/sars_cov_2_phylogeny.

[15] http://emboss.sourceforge.net/apps/release/6.6/embassy/phylipnew/fdnadist.html.

[16] http://emboss.sourceforge.net/apps/release/6.6/embassy/phylipnew/fneighbor.html.

[17] https://github.com/tommaso-green/clustering-sarscov2-phylogenies.

[18] https://nextstrain.org/ncov/global?l=clock.

[19] https://nextstrain.org/ncov/global.

[20] https://www.livescience.com/coronavirus-mutation-rate.html.

[21] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.SpectralClustering.html.

[22] . Severe acute respiratory syndrome coronavirus 2 isolate wuhan-hu-1. https://www.ncbi.nlm.nih.gov/nuccore/MN908947.3, 2020.

[23] Severe acute respiratory syndrome coronavirus 2 isolate wuhan-hu-1, complete genome, 2020. https://www.ncbi.nlm.nih.gov/nuccore/NC_045512.

[24] Almeida, J. S., and Vinga, S. Rényi continuous entropy of dna sequences.

[25] Almeida, J. S., and Vinga, S. Universal sequence map (usm) of arbitrary discrete sequences. https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-3-6, 2002.

[26] Almeida, J. S., and Vinga, S. Local renyi entropic profiles of dna sequences. https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-8-393, 2007. BMC Bioinformatics.

[27] Balaban, M., Moshiri, N., Mai, U., Jia, X., and Mirarab, S. Treecluster: Clustering biological sequences using phylogenetic trees. *PloS one 14*, 8 (2019).

[28] Caramelli, D. *Antropologia Molecolare, manuale di base.* Firenze University Press, 2009.

[29] Cock, P. J. A., Antao, T., Chang, J. T., Chapman, B. A., Cox, C. J., Dalke, A., Friedberg, I., Hamelryck, T., Kauff, F., Wilczynski, B., and de Hoon, M. J. L. Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics 25*, 11 (03 2009), 1422–1423.

[30] Comin, M. Ep-sim: Multiple-resolution alignment-free measure based on entropic profiles. http://www.dei.unipd.it/~ciompin/main/EP-sim.html, 2016.

[31] Comin, M., and Morris, A. On the comparison of regulatory sequences with multiple resolution entropic profiles. https://doi.org/10.1186/s12859-016-0980-2, 2016.

[32] Comin, M., and Morris, A. Fast computation of entropic profiles for the detection of conservation in genomes. http://www.dei.unipd.it/~ciompin/papers/PRIB-Proceedings.pdf, 2020.

[33] Gascuel, O., and Steel, M. Neighbor-joining revealed. molecular biology and evolution. https://academic.oup.com/mbe/article/23/11/1997/1322446, 2006.

[34] Gotoh, O. An improved algorithm for matching biological sequences. *Journal of molecular biology 162*, 3 (December 1982), 705—708.

[35] Hirschberg, D. S. A linear space algorithm for computing maximal common subsequences. *Commun. ACM 18*, 6 (June 1975), 341–343.

[36] Holland, L. A., Kaelin, E. A., Maqsood, R., Estifanos, B., Wu, L. I., Varsani, A., Halden, R. U., Hogue, B. G., Scotch, M., and Lim, E. S. An 81-nucleotide deletion in sars-cov-2 orf7a identified from sentinel surveillance in arizona (january to march 2020). *Journal of Virology 94*, 14 (2020).

[37] Jones, N. C., and Pevzner, P. *Introduction to Bioinformatics Algorithms*. MIT Press, 2014.

[38] Kurtzman, C. P., Fell, J. W., and Boekhout, T. The yeasts. https://www.sciencedirect.com/science/article/pii/B9780444521491000124?via%3Dihub, 2010.

[39] Moore, R. M., O., H. A., W., M. S. M. P. S., and Eric, W. K. Iroki: automatic customization and visualization of phylogenetic trees. https://doi.org/10.7717/peerj.8584, 2020.

[40] Myers, E. W., and Miller, W. Optimal alignments in linear space. *Bioinformatics 4*, 1 (03 1988), 11–17.

[41] Ogilvie, H. A. Tree comparisons. https://www.cs.rice.edu/~ogilvie/comp571/2018/10/18/tree-metrics.html, 2018.

[42] Raffaele, U. V.-S. S. Viaggio al centro del virus: com'è fatto sars-cov-2, 2020. https://www.unisr.it/news/2020/3/viaggio-al-centro-del-virus-come-e-fatto-sars-cov-2.

[43] Reinert, G., Chew, D., Sun, F., and Waterman, M. S. Alignment-free sequence comparison (i): Statistics and power. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2818754, 2009.

[44] Rice, P. Longden, I., and Bleasby, A. EMBOSS: The European Molecular Biology Open Software Suite. *Trends in Genetics 16*, 6 (2000), 276–277.

[45] Shalev-Shwartz, S., and Ben-David, S. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[46] The New Yorks Times. How coronavirus mutates and spreads, 2020. https://www.nytimes.com/interactive/2020/04/30/science/coronavirus-mutations.html.

[47] University of Washington. Phylip documentation. http://evolution.genetics.washington.edu/phylip.html.

[48] University of Washington. Phylip documentation: neighbor. http://evolution.genetics.washington.edu/phylip/doc/neighbor.html.

[49] Wikipedia. Upgma. https://en.wikipedia.org/w/index.php?title=UPGMA&oldid=950977468, 2020.

[50] Wilbur, W. J., and Lipman, D. J. Rapid similarity searches of nucleic acid and protein data banks, 1983. https://pdfs.semanticscholar.org/cb83/3f4d8c34f7269b907259b0cf6554e5524643.pdf.

[51] Zielezinski, A., Vinga, S., Almeida, J., and Karlowski, W. M. Alignment-free sequence comparison: benefits, applications, and tools. https://www.ncbi.nlm.nih.gov/pubmed/28974235, 2017.