Multiple Sequence Alignment (MSA) di sequenze SARS-CoV-2

Edoardo Silva 816560
Davide Marchetti 815990

A.A.: 2019/2020

1 Abstract

Date le variazioni sulle sequenze rilevate e catalogate nelle parti precedenti, costruiremo appositamente una matrice binaria di caratteri, evitando la generazione della "matrice proibita", da utilizzare per la realizzazione della filogenesi perfetta delle sequenze.

A partire dai file di output della prima parte del progetto, l'algoritmo genera una matrice binaria utilizzando come indici di riga gli identificativi delle sequenze e come colonne un identificativo univoco assegnato ad ogni variazione per poi estrarne la più grande matrice adatta alla generazione di un albero di una filogenesi perfetta.

Successivamente realizzeremo l'albero filogenetico con l'ausilio delle librerie **Bio** e **anytree**, confronteremo dell'albero ottenuto rispetto a quello prodotto con i tool di allineamento e analizzeremo i risultati ottenuti in relazione alle ipotesi avanzate dal report preliminare della prima parte di progetto per capire come il virus si sia introdotto nelle aree geografiche esaminate.

2 Albero filogenetico

Un albero filogenetico è un albero che descrive la sequenze di eventi di speciazione che hanno portato alle specie attuali, dove con il termine speciazione si intende la differenziazione in due gruppi dove la differenza genica è predominante.

Il `principio di Parsimonia` sancisce come ogni specie sia specificata da un insieme di caratteri o attributi. A livello genomico, un carattere o attributo può assumere diversi significati:

- ordine o composizione dei geni in un genoma
- composizione delle proteine
- cambiamenti genomici rari (RGC)
- acquisizione o perdita di introni

La filogenesi perfetta deriva dall'unione di due principi:

- `Dollo parsimony`: un carattere può essere acquisito una sola volta ($0 \to 1$), ma può venire perso più volte ($1 \to 0$).
- `Camin-Sokal parsimony`: un carattere può essere acquisito molteplici volte ($0 \to 1$), ma non può essere perso.

In una filogenesi perfetta risulta possibile quindi acquisire e perdere un carattere esclusivamente una volta. Da questa affermazione ne deriva che un albero rappresentante una filogenesi perfetta avrà alcune caratteristiche:

1. Ogni carattere identifica esattamente una ramificazione.
2. L'albero ha il numero di foglie equivalente al numero di campioni analizzati, con ogni campione come foglia.
3. Seguendo il percorso da una foglia fino alla radice si identificano i caratteri presenti nella sequenza posta sulla foglia di partenza.

3 Algoritmo

3.1 Matrice delle variazioni

Per la costruzione dell'albero abbiamo scelto di utilizzare ciascuna alterazione come identificativa di un singolo carattere, associandovi un codice univoco. La matrice così ottenuta pone sulle righe l'identificativo di ciascuna sequenza analizzata (indice i) e sulle colonne, denominate caratteri, le singole variazioni (indice j).

Ciascuna cella della matrice viene popolata con il valore `1` qualora il j-esimo carattere (variazione) si manifesti nell'i-esima sequenza, altrimenti viene inserito il valore zero.

Procedendo al riempimento della matrice in questo modo si ottiene la matrice di caratteri riportata in fig. 1.

Sequence	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22	C23	C24	C25	C26	C27	C28	C29	C30	C31	C32	C33	C34	C35	C36	C37	C38	C39	C40	C41	C42	C43	C44	C45	C46	C47	C48	C49	C50	C51	C52
NC_045512.2	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	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
EPI_ISL_442523	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EPI_ISL_437512	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	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
EPI_ISL_447469	0	0	0	0	0	0	0	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EPI_ISL_417444	0	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	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0
EPI_ISL_437334	0	1	1	0	0	1	0	0	0	0	0	1	1	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	0	0	0	0
EPI_ISL_437317	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
MT320891.2	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MT281530.2	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MT276598.1	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MT276597.1	0	0	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
MT240479.1	0	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	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0
MT262993.1	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0
MT327745.1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0

Figura 1: Matrice di caratteri iniziale

3.2 Filogenesi perfetta

Tuttavia, prima di procedere con la creazione dell'albero è necessario verificare che la matrice binaria generata in precedenza sia valida per una filogenesi perfetta, cioè che non sia contenuta una sottomatrice che sia la `"matrice proibita"`.

Applicando l'algoritmo nel listato 3 alla matrice binaria completa di tutti i caratteri, si ottiene la matrice illustrata in fig. 2. Durante il procedimento vengono esclusi i caratteri: `C3, C7, C12, C31` e `C44`.

3.3 Generazione dell'albero

A questo punto è possibile procedere con la ricostruzione dell'albero filogenetico a partire dalla `candidate_matrix` e utilizzando funzioni definite nel file `phylogeny.py` per costruire e visualizzare in output l'albero della filogenesi.

Sequence	C2	C6	C1	C4	C5	C20	C10	C9	C40	C41	C42	C43	C8	C23	C11	C13	C14	C15	C16	C17	C18	C19	C21	C22	C52	C24	C25	C50	C49	C48	C47	C46	C45	C39	C38	C37	C36	C35	C34	C33	C32	C30	C29	C28	C27	C51	C26
NC_045512.2	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EPI_ISL_442523	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EPI_ISL_437512	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1
EPI_ISL_447469	0	0	0	0	0	0	1	1	0	0	0	0	1	0	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	0	0	0	0	0	0
EPI_ISL_417444	1	0	0	0	0	0	0	0	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	0	0	0	0	0	0	0	0	0
EPI_ISL_437334	1	1	0	0	0	1	0	0	0	0	0	0	0	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
EPI_ISL_437317	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	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
MT320891.2	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MT281530.2	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MT276598.1	0	0	0	0	0	0	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	0	0	0	1	1	1	1	0	0	0	0	0	0
MT276597.1	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	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MT240479.1	1	0	0	0	0	0	0	0	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	0	0	0	0	0	0	0	0	0
MT262993.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
MT327745.1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura 2: Matrice di caratteri per filogenesi perfetta

3.3.1 Creazione dell'albero

La generazione dell'albero viene svolta come segue:

1. Ordinamento decrescente delle colonne della matrice in base al numero di valori pari a `1` presenti.

2. A partire dal nodo `root`, per ogni sequenza genera una sequenza di nodi uno per ogni variazione presente, collegandola alla variazione precedente o al nodo `root`.

   Durante l'elaborazione delle sequenze successive, se esiste già un figlio del `current_node` rappresentante la variazione attuale, questo viene riutilizzato, altrimenti viene creato un nuovo nodo e collegato come figlio di `current_node`.

3. Conversione dell'albero precedentemente generato in un `newick_tree` elaborabile dalla libreria `biopython.Phylo` usata per la visualizzazione.

4. Inserimento delle informazioni aggiuntive sulle sequenze nelle foglie dell'albero. Questo step non può essere effettuato durante la conversione a `newick_tree` perché alcuni caratteri usati verrebbero interpretati come parte della struttura dell'albero, alterando il risultato finale.

5. Salvataggio e visualizzazione dell'albero filogenetico finale.

4 Analisi dei risultati e conclusioni

Oltre alla generazione dell'albero tramite il nostro script, sono stati prodotti due diversi alberi filogenetici a partire dagli allineamenti: in fig. 3 é illustrato l'albero generato con i parametri di default, mentre in fig. 4 quello ottenuto abilitando il parametro `Exclude Gaps=On`.

È necessario sottolineare che si è scelto di riportare esclusivamente gli alberi generati dall'allineamento effettuato con ClustalW poiché identici a quelli prodotti da MUSCLE.

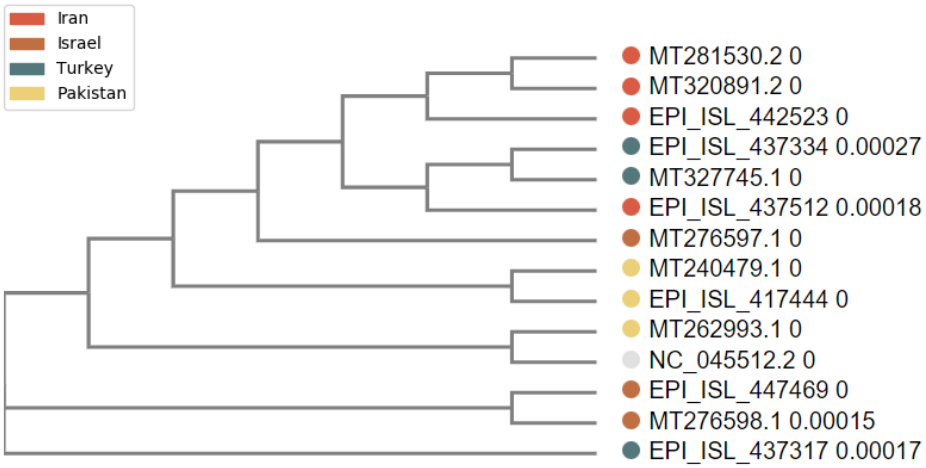

Figura 3: Albero generato con parametri di default

La differenza così evidente di posizionamento della reference tra l'albero in fig. 3 e quelli in fig. 4 avviene in quanto il parametro `Exclude Gaps` ignora le colonne per le quali in almeno una delle sequenze si verifica una cancellazione (gap), forzando l'utilizzo esclusivamente delle colonne dove si ha una base in tutte le sequenze.

Fin dalla prima fase del progetto abbiamo osservato come le cancellazioni avvengano agli estremi delle sequenze. Successivamente nella seconda parte del progetto abbiamo avuto conferma di quanto ipotizzato: nella quasi totalità dei casi le variazioni che coinvolgono geni sono sostituzioni.

Questo fornisce una possibile spiegazione sul perchè l'esclusione dei gap determini uno spostamento così marcato di alcune sequenze.

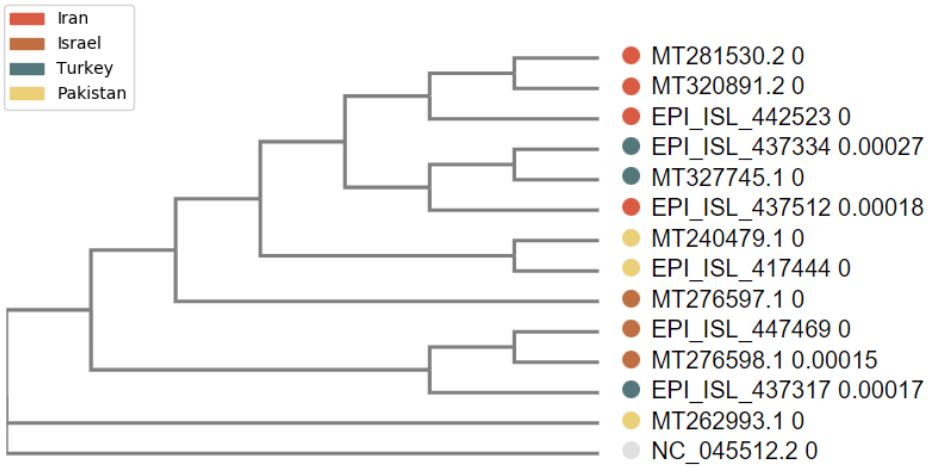

Figura 4: Albero generato con `Exclude Gaps=On`

Confrontando ora gli alberi generati dall'allineamento con ClustalW (fig. 3 e 4) e quello generato dal nostro script (fig. 5), si riesce ad identificare un gruppo di sequenze in comune che copre le sequenze tra `MT281530.2` e `EPI_ISL_437512`.

Osservando in dettaglio i tre alberi si possono notare alcuni dettagli interessanti:

- Il nodo israeliano etichettato con `MT276597.1`, includendo i gap presenta mutazioni differenti rispetto alle altre due sequenze israeliane. Questo potrebbe far presupporre che derivi da un ceppo diverso del virus o sia entrato nel paese da un'area geografica differente.
- La sequenza pakistana `MT262993.1` presenta alterazioni differenti rispetto alle altre sequenze dello stesso paese. Come già osservato nell'analisi iniziale questa sequenza presenta esclusivamente cancellazioni e nessuna sostituzione.
- I rami restanti degli alberi nelle fig. 4 e 5 illustrano come nella stessa area geografica i genomi siano simili tra loro e condividano molte variazioni il che ci porta a ipotizzare che la diffusione all'interno del paese si sia sviluppata a partire da un singolo paziente per nazione e non da pazienti contagiati da mutazioni diverse del virus.

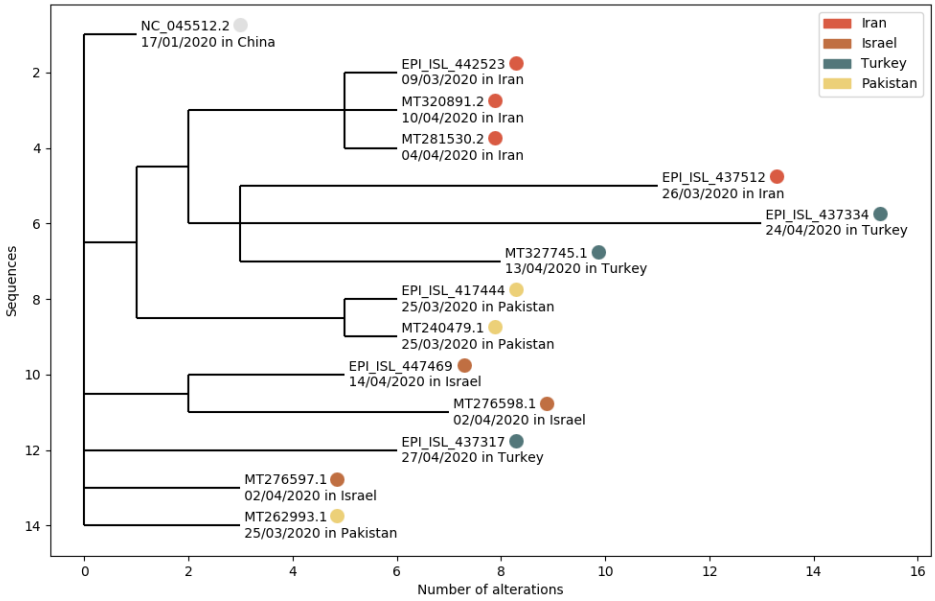

Figura 5: Albero di output del nostro script

- I sequenziamenti rilevati in `Israele` sembrano avere una possibile origine differente e quasi non condividere mutazioni simili con le sequenze degli altri paesi analizzati.

- I ceppi iraniani e turchi presentano mutazioni comuni e le sequenze di entrambi i paesi pare condividano una variazione con alcune sequenze rilevate in Pakistan.

In conclusione, gli alberi nelle fig. da 3 a 5 risultano abbastanza simili, tuttavia presentano alcune differenze nel posizionamento di alcune sequenze.

Ipotizziamo che queste derivino principalmente dal fatto che l'albero in fig. 5 è costruito basandosi sul numero di alterazioni, mentre gli alberi in fig. 3 e 4 vengono generati rispetto alle intere sequenze.

4.1 Divisone del lavoro

Durante la realizzazione del progetto entrambi i componenti del gruppo hanno partecipato attivamente alla sua realizzazione. In particolare:

- **Edoardo Silva** si è occupato principalmente di recuperare i file esterni da elaborare e confrontare.
- **Davide Marchetti** si è occupato principalmente di generare i file di output.
- Entrambi hanno lavorato alla creazione ed elaborazione della matrice e dell'albero, con tutte le funzioni ausiliarie allo scopo del progetto.

5 Listati di codice

Listing 1: Caricamento dei file necessari per l'elaborazione

```
reference_id = load_fasta_id(
  os.path.join('..', '..', 'project-1', 'input', 'reference.fasta')
)
sequence_ids = read_sequence_ids(paths=[
  os.path.join('..', '..', 'project-1', 'input', 'GISAID'),
  os.path.join('..', '..', 'project-1', 'input', 'ncbi'),
])
sequence_ids.insert(0, reference_id) #insert reference no variations

clustal_output = load_output('Clustal-NC_045512.2.json')
variations = clustal_output['unmatches'].items()
```

Listing 2: Generazione della matrice binaria di caratteri

```
for key, value in variations:
  row = np.zeros(len(sequence_ids))
  indexes.append('C{}'.format(counter))
  for sequence in value['sequences']:
    row[sequence_ids.index(sequence)] = 1
  rows.append(row)
  counter += 1

trait_matrix = pd.DataFrame(rows, index=indexes, columns=
    ↪ sequence_ids, dtype=np.uint8).transpose()
trait_matrix = phylogeny.reorder_columns(trait_matrix, axis=0)
trait_matrix.to_csv(os.path.join('..', 'output', 'character_table.
    ↪ csv'))
```

Listing 3: Funzione di generazione della matrice binaria per filogenesi perfetta

```
def get_perfect_phylogeny_character_matrix(df):
  columns = df.columns
  candidate_matrix = df[columns[0:1]]
  for i in range(1, len(columns)):
    candidate_matrix = candidate_matrix.join(df[columns[i:i+1]])

    if phylogeny.is_forbidden_matrix(candidate_matrix):
      candidate_matrix = candidate_matrix.drop(labels=
          ↪ candidate_matrix.columns[-1], axis=1)

  return candidate_matrix
```

Listing 4: Funzione di creazione dell'albero

```
root = Node('root', edges={})
for i, row in df.iterrows():
  current_node = root

  for j in range(len(row)):
    # If alteration is present in the current sequence
    if row.iloc[j]:
      # If current_node has a link to the variation with label=j
      if j in current_node.edges:
        # Follow the same path without creating new nodes
        current_node = current_node.edges[j]
      else:
        u = Node('U-{}'.format(row.index[j]), edges={})
        current_node.parent = current_node
        current_node.edges[j] = u
        current_node = u

  Node(i, parent=current_node)
```

Listing 5: Funzione di conversione in Newick Tree

```
def to_newick_tree(node):
  if node.is_leaf:
    return node.name
  return '({})'.format(','.join([ to_newick_tree(child) for child in
      ↪  node.children ]))

newick_string = to_newick_tree(root)
tree = Phylo.read(io.StringIO(newick_string), 'newick')
```

Listing 6: Funzione per l'inserimento dei dati nelle foglie dell'albero

```
def merge_sequences_data(node, sequences_data):
  if node.is_terminal():
    data = sequences_data[node.name]
    node.name = '{}\n {} in {}'.format(node.name, data['date'], data
        ↪ ['location'])
    return

  [ merge_sequences_data(child, sequences_data) for child in node.
      ↪ clades ]

root = newick_tree.clade
merge_sequences_data(root, sequences_data)
```