

# 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 una matrice binaria di caratteri.

In seguito costruire la filogenesi perfetta delle sequenze, rimuovendo le colonne che genererebbero la matrice proibita, e successivamente confrontare l'albero ottenuto con quello ottenibile con i tools di allineamento.

Descriveremo infine i risultati ottenuti.

## 2 Algoritmo

### 2.1 Matrice delle variazioni

Inizialmente, vengono recuperati gli identificativi delle 14 sequenze usate e caricati i file delle variazioni degli allineamenti generati in output nella prima parte del progetto:

**Listing 1:** Caricamento dei file necessari per l'elaborazione

```
1 reference_id = load_fasta_id(  
2     os.path.join('..', '..', 'project-1', 'input', 'reference.fasta')  
3 )  
4 sequence_ids = read_sequence_ids(paths=[  
5     os.path.join('..', '..', 'project-1', 'input', 'GISAID'),  
6     os.path.join('..', '..', 'project-1', 'input', 'ncbi'),  
7 ])  
8 sequence_ids.insert(0, reference_id) #insert reference no variations  
9  
10 clustal_output = load_output('Clustal-NC_045512.2.json')  
11 variations = clustal_output['unmatches'].items()
```

L'algoritmo a partire dai file di output della prima parte del progetto genera una matrice binaria utilizzando come indici di riga gli identificativi delle sequenze e come colonne un identificativo univoco assegnato ad ogni variazione.

La matrice binaria così impostata contiene il valore 1 qualora la variazione identificata dalla colonna sia presente nella sequenza identificata dalla riga, altrimenti il valore della cella sarà pari a 0. Questa è salvata nel file `character_table.csv`.

**Listing 2:** Generazione della matrice binaria di caratteri

```
1 for key, value in variations:  
2     row = np.zeros(len(sequence_ids))  
3     indexes.append('C{}'.format(counter))  
4     for sequence in value['sequences']:  
5         row[sequence_ids.index(sequence)] = 1  
6     rows.append(row)  
7     counter += 1  
8  
9 trait_matrix = pd.DataFrame(rows, index=indexes, columns=  
10     ↪ sequence_ids, dtype=np.uint8).transpose()  
11 trait_matrix = phylogeny.reorder_columns(trait_matrix, axis=0)  
12 trait_matrix.to_csv(os.path.join('..', 'output', 'character_table.  
    ↪ csv'))
```

## 2.2 Filogenesi perfetta

Prima di procedere con la creazione dell'albero è necessario verificare che la matrice binaria generata in precedenza sia valida per una filogenesi perfetta.

Il metodo riportato nel listato 3 riporta il codice utilizzato per costruire la matrice binaria più grande che non sia "matrice proibita".

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

```
1 def get_perfect_phylogeny_character_matrix(df):
2     columns = df.columns
3     candidate_matrix = df[columns[0:1]]
4     for i in range(1, len(columns)):
5         candidate_matrix = candidate_matrix.join(df[columns[i:i+1]])
6
7     if phylogeny.is_forbidden_matrix(candidate_matrix):
8         candidate_matrix = candidate_matrix.drop(labels=
9             ↪ candidate_matrix.columns[-1], axis=1)
10    return candidate_matrix
```

## 2.3 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.

**Listing 4:** Chiamata alla generazione dell'albero

```
1 candidate_matrix = get_perfect_phylogeny_character_matrix(
2     ↪ trait_matrix)
3 phylogeny.build_tree(candidate_matrix)
```

### 2.3.1 Creazione dell'albero

La generazione dell'albero viene svolta come segue:

1. Ordinamento decrescente delle colonne del dataframe in base al numero di valori 1 presenti.

**Listing 5:** Ordinamento decrescente delle colonne

```
1 sorted_axis = df.sum(axis=0).sort_values(ascending=False)
2 return df[sorted_axis.index]
```

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`.

**Listing 6:** Funzione di creazione dell'albero

```
1 root = Node('root', edges={})
2 for i, row in df.iterrows():
3     current_node = root
4
5     for j in range(len(row)):
6         # If alteration is present in the current sequence
7         if row.iloc[j]:
8             # If current_node has a link to the variation with label=j
9             if j in current_node.edges:
10                 # Follow the same path without creating new nodes
11                 current_node = current_node.edges[j]
12             else:
13                 u = Node('U-{}'.format(row.index[j]), edges={})
14                 current_node.parent = current_node
15                 current_node.edges[j] = u
16                 current_node = u
17
18     Node(i, parent=current_node)
```

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

**Listing 7:** Funzione di conversione in Newick Tree

```
1 def to_newick_tree(node):
2     if node.is_leaf:
3         return node.name
4
5     return '({})'.format(','.join([ to_newick_tree(child) for child in
6         ↪ node.children ]))
7
8 newick_string = to_newick_tree(root)
9 tree = Phylo.read(io.StringIO(newick_string), 'newick')
```

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.

**Listing 8:** Funzione per l'inserimento dei dati nelle foglie dell'albero

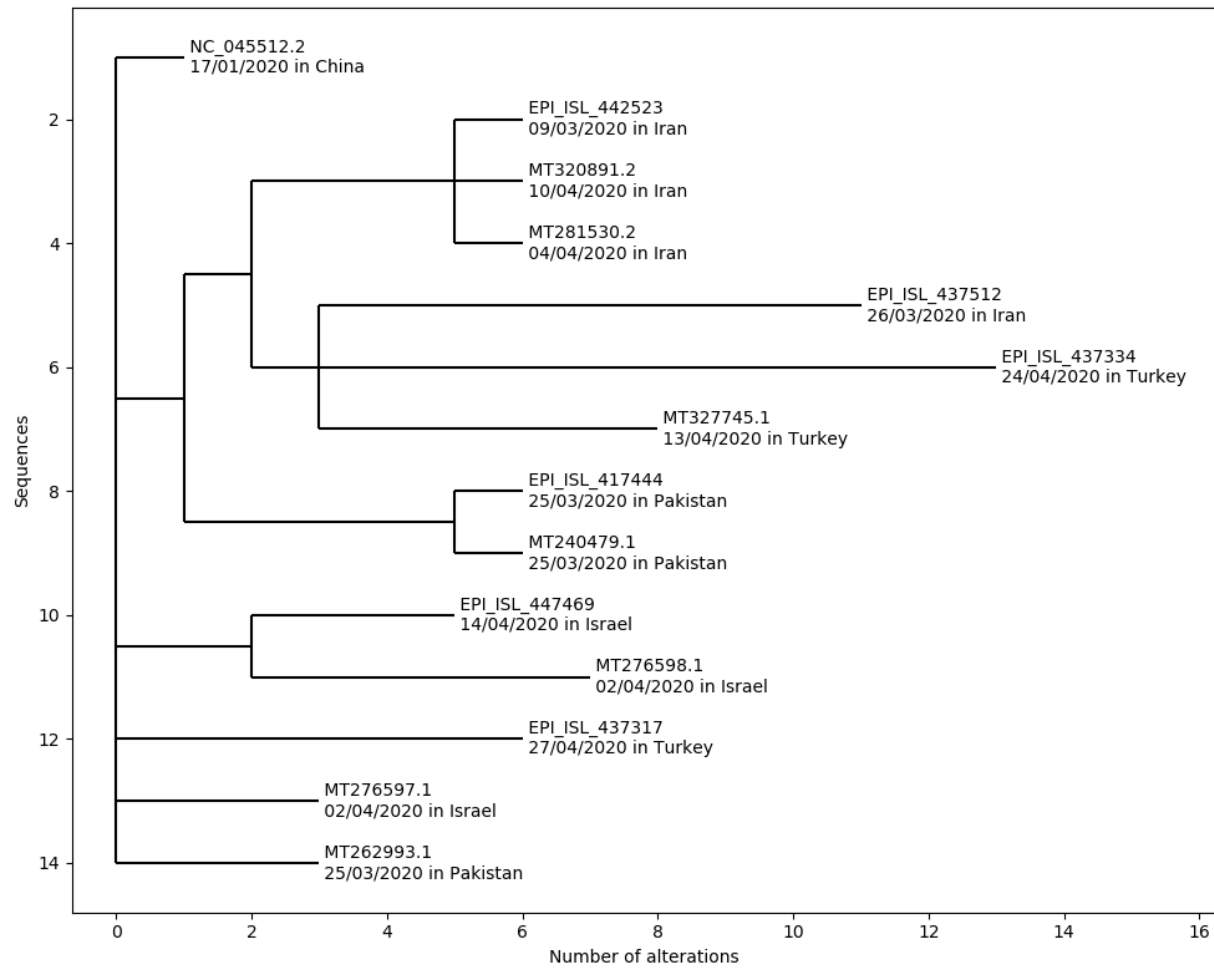
```
1 def merge_sequences_data(node, sequences_data):
2     if node.is_terminal():
3         data = sequences_data[node.name]
4         node.name = '{}\n {} in {}'.format(node.name, data['date'], data
           ↪ ['location'])
5     return
6
7     [ merge_sequences_data(child, sequences_data) for child in node.
           ↪ clades ]
8
9 root = newick_tree.clade
10 merge_sequences_data(root, sequences_data)
```

5. Salvataggio e visualizzazione dell'albero filogenetico finale.

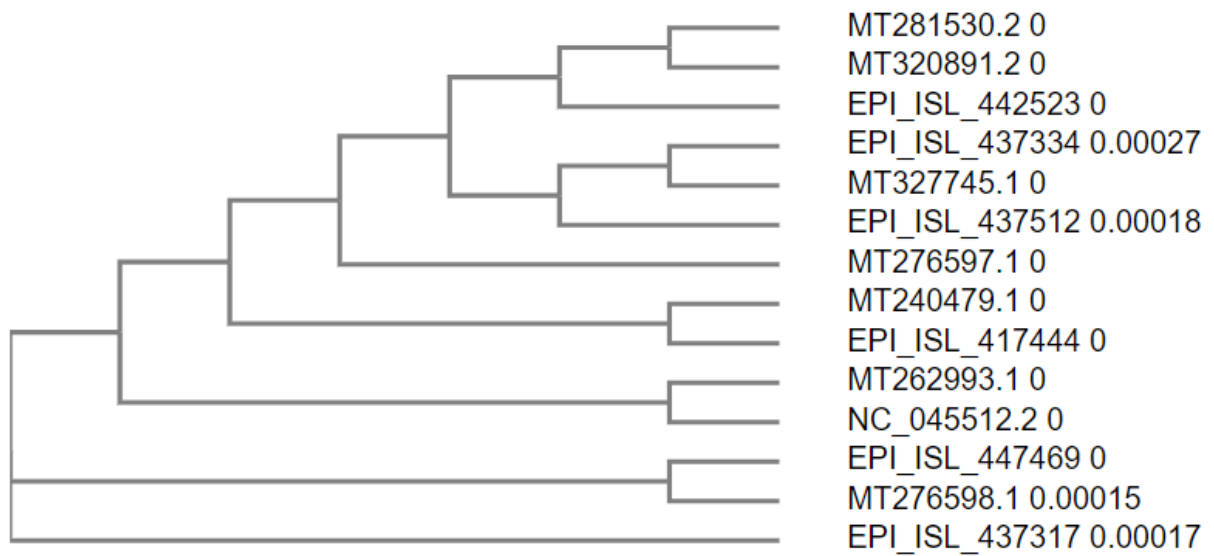
**Listing 9:** Salvataggio e visualizzazione dell'albero filogenetico finale

```
1 fig = plt.figure(figsize=(10, 8))
2 ax = fig.add_subplot(1, 1, 1)
3
4 Phylo.draw(tree, do_show=False, axes=ax)
5
6 ax.set_xlabel('Number of alterations')
7 ax.set_ylabel('Sequences')
8 plt.tight_layout()
9 plt.savefig(os.path.join '..', 'output', 'phylogenetic-tree.png'))
10 plt.show()
```

### 3 Confronto alberi



**Figura 1:** Albero di output dello script



**Figura 2:** Albero di output del sito delle sequenze

## 4 Conclusioni

Come è possibile notare i 2 alberi ottenuti hanno delle differenze:

- nell'albero dei tools, in caso di 3 alberi con gli stessi disallineamenti, riesce a controllare i 2 più vicini, non badando alla filogenesi perfetta ma sfruttando ogni allineamento.
- il nodo MT276597 israeliano ha molti disallineamenti scartati nel nostro codice ed in entrambi gli alberi è distante dalle altre 2 sequenze israeliane (possibilmente proveniente da un'altra zona geografica? ondata di ritorno?)

La quasi totalità dei nodi mostra come nella stessa area geografica i genomi siano simili tra loro, come appartenenti ad un singolo nuovo ceppo per nazione.

I ceppi iraniani e turchi presentano quasi le stesse mutazioni, e ne condividono 1 con i ceppi Pakistani, al contrario il ceppo di virus rilevato in Israele sembra provenire da un percorso diverso (richiesti nuovi studi in questo ambito).

In conclusione, i 2 alberi sono abbastanza simili ad indicare che non vi siano errori nel codice, solo diverse scelte algoritmico-implementative.