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

Edoardo Silva 816560
Davide Marchetti 815990

A.A.: 2019/2020

1 Abstract

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

```
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()
```

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

```
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'))
```

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=
             ↪ candidate_matrix.columns[-1], axis=1)
9
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(
       ↪ trait_matrix)
2  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

```
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)
```

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

```
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')
```

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

```
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)
```

5. Salvataggio e visualizzazione dell'albero filogenetico finale.

Listing 9: Salvataggio e visualizzazione dell'albero filogenetico finale

```
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(1, 1, 1)

Phylo.draw(tree, do_show=False, axes=ax)

ax.set_xlabel('Number of alterations')
ax.set_ylabel('Sequences')
plt.tight_layout()
plt.savefig(os.path.join('..', 'output', 'phylogenetic-tree.png'))
plt.show()
```

3 Confronto alberi

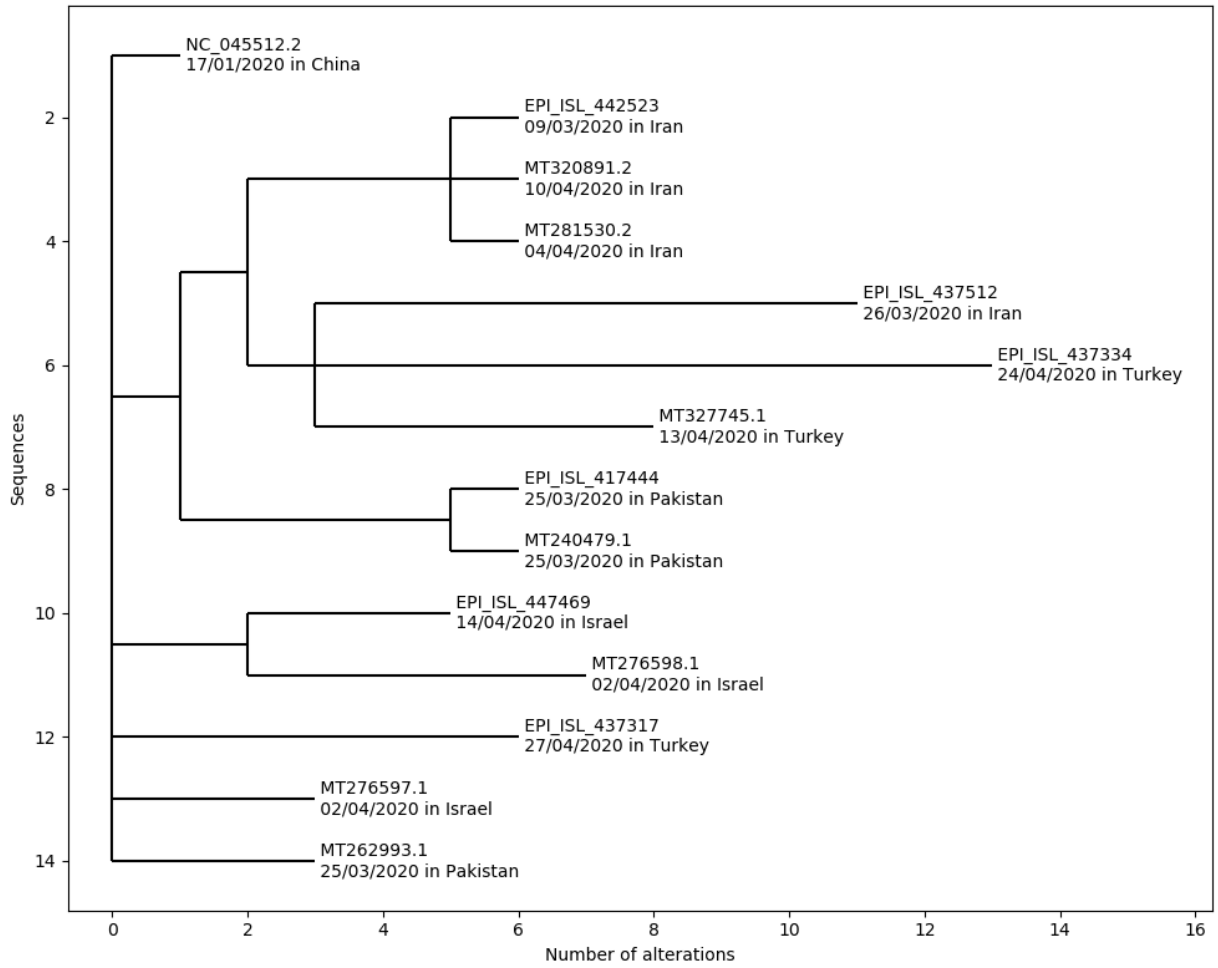

Figura 1: Albero di output dello script

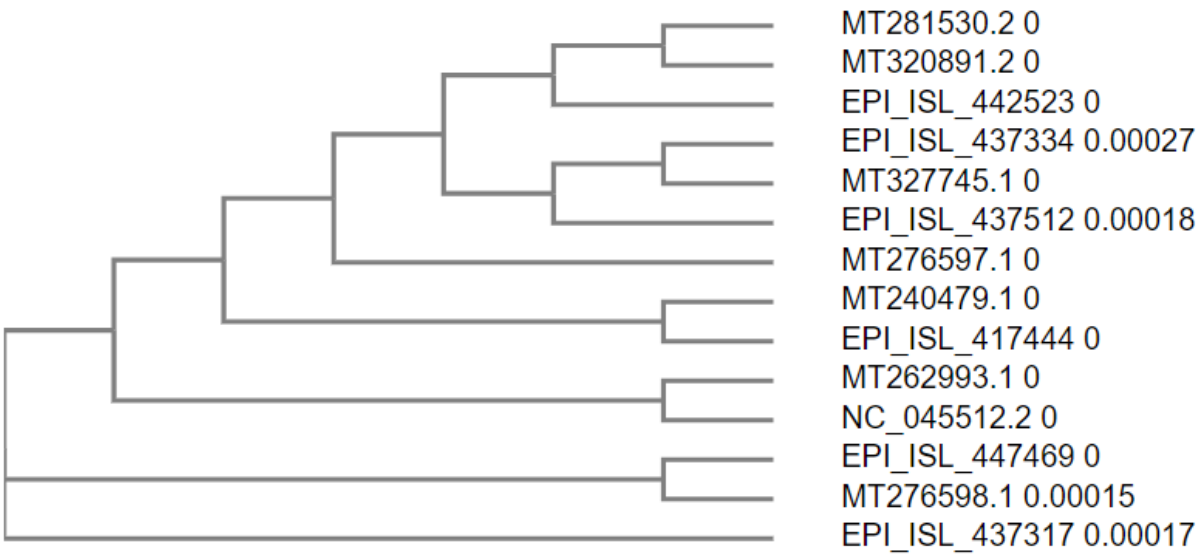

Figura 2: Albero di output del sito delle sequenze

4 Conclusioni