Practical 2: Perfect Phylogeny

Ximena Moure Eliya Tiram

19/09/2023 , submission deadline 25/09/2023

1. **Extract Segregating Sites From sequences.fa Into a Binary Matrix**
   The following code reads the file given in the exercise and answer the questions within it.

```python
from collections import Counter

def read_sequences(filename):
    sequences = []
    with open(filename, 'r') as file:
        sequence = ""
        for line in file:
            if line.startswith('>'):
                if sequence:
                    sequences.append(sequence)
                sequence = ""
            else:
                sequence += line.strip()
        if sequence:
            sequences.append(sequence)
    return sequences

def most_frequent_nucleotide(nucleotides):
    element_counts = Counter(nucleotides)
    most_common_elements = element_counts.most_common(1)
    most_common_element, count = most_common_elements[0]
    return most_common_element

def segregating_sites(sequences):
    reference = sequences[0]
    matrix = []

    for position in range(len(reference)):
        nucleotides = [seq[position] for seq in sequences]
        unique_nucleotides = set(nucleotides)
        if len(unique_nucleotides) > 1 and '.' not in unique_nucleotides and 'N' not in unique_nucleotides:
            column = []
            ref_nucleotide = most_frequent_nucleotide(nucleotides)
            for nucleotide in nucleotides:
                column.append(0 if nucleotide == ref_nucleotide else 1)
            matrix.append(column)

    matrix = list(map(list, zip(*matrix)))
    matrix = remove_duplicate_columns(matrix)
    return matrix

def remove_duplicate_columns(matrix):
    # Transpose the matrix to treat columns as rows
    transposed = list(map(list, zip(*matrix)))
    seen = set()
    unique_transposed = []
    for col in transposed:
        col_tuple = tuple(col)
        if col_tuple not in seen:
            unique_transposed.append(col)
            seen.add(col_tuple)
    # Transpose back to the original form
    return list(map(list, zip(*unique_transposed)))
```

```python
def main():
    filename = 'sequences.fa'
    sequences = read_sequences(filename)
    print("Sequences length", len(sequences))
    matrix = segregating_sites(sequences)
    num_segregating_sites = len(matrix[0])
    print(f"There are {num_segregating_sites} segregating sites.")
    # for row in matrix:
    #     print(''.join(map(str, row)))
    if (has_perfect_phylogeny(matrix)):
        print(f"Perfect phylogeny")
    else:
        print("Not perfect phylogeny")
if __name__ == '__main__':
    main()

```

Listing 1: Python code ex1

2. **How many genomic sequences are there?**
   There are 11 genomic sequences
3. **How many segregating sites do they have?**
   There are 19 segregating sites.
4. **Determine whether there is a perfect phylogeny**
   The following code is an addition to the code above in order to answer the following sections.
   There is not a perfect phylogeny for the segregating sites of the sequences.

```python
def sort_columns(matrix):
    # Transpose the matrix to get columns as rows
    transposed = list(map(list, zip(*matrix)))
    # Sort the columns (which are now rows) based on their binary value
    sorted_transposed
    = sorted(transposed, key=lambda x: [int(i) for i in x], reverse=True)
    # Transpose back to get the sorted matrix
    for row in list(map(list, zip(*sorted_transposed))):
        print(''.join(map(str, row)))
    return list(map(list, zip(*sorted_transposed)))

def has_perfect_phylogeny(matrix):
    matrix = sort_columns(matrix)
    # Iterate over each pair of columns
    for i in range(len(matrix[0])):
        for j in range(i + 1, len(matrix[0])):
            Oi = {row for row, val in enumerate(matrix) if val[i] == 1}
            Oj = {row for row, val in enumerate(matrix) if val[j] == 1}

            # Check the conditions
            if not (Oi.isdisjoint(Oj) or Oi.issubset(Oj) or Oj.issubset(Oi)):
                return False
    return True
```

Listing 2: Python code ex4

5. **What is the running time of your script, as a function of the number n of genomic sequences and the number m of segregating sites?**
   $O(nm^2)$
6. **What is the best possible running time of an algorithm to solve the perfect phylogeny problem?**
   According to the lecture the best running time for the algorithm asked is O(nm)