# Python program for alignment free phylogenomics

Lucas van Dijk

July 20, 2015

## 1 Introduction

Using alignment-free methods to calculate the distances between sequences has gotten quite a bit of attention recently. In an alignment-free method, shared properties of the sequences (such as number of exact or inexact shared k-length subsequences) are the basis for the distance.

One of the possible applications is constructing phylogenetic trees, where multiple sequence alignment has been the standard for calculating the distances of the sequences. However, multiple sequence alignment is an NP-complete problem [1], and thus does not scale very well with larger sequences or more species.

In [2] Chat et al. used several variants of the $D_2$ statistic (number of shared k-mers) to construct phylogenetic trees and showed that alignment-free methods definitely have potential for large scale phylogenomics.

In this report we will discuss a Python implementation for constructing phylogenetic trees using the $D_2$ and $D_2$-neighbourhood statistics. For evaluation, we compare the generates trees using the state of the art multiple alignment tool MAFFT [3].

## 2 Method

As mentioned before, our Python program can use the $D_2$ and $D_2$-neighbourhood statistic to construct phylogenetic trees [2]. We summarize the algorithm below:

1. Construct distance matrix between the sequences, using the statistic of choice.
2. Construct the tree using neighbour-joining.

The program has a few statistics built in:

- $D_2$, which is based on the number of shared k-mers
- $D_2^N$ ($D_2$-neighbourhood), which also takes the neighbourhood ($n$ point mutations of a k-mer) into account.

## $D_2$ statistic

The $D_2$ statistic is defined as follows:

$$D_2 = \sum_{w \in A^k} X_w Y_w$$

Where $A^k$ is the set of all possible k-length words using alphabet $A$, $X_w$ is the word count of word $w$ in sequence $X$, and $Y_w$ is the word count of word $w$ in sequence $Y$.

This statistic is covered in more detail in Song et al. [4].

## $D_2^n$, $D_2$ with neighbourhood

The $D_2$-neighbourhood statistic, or $D_2^n$, also takes the *neighbourhood* of each k-length word $w$ into account. The neighbourhood of a word $w$ is defined as alle possible $n$ point mutations of the word $w$. For example, when $w$ = AAA ($k = 3$), and $n = 1$, then the neighbourhood of $w$ consists of: AAA, AAC, AAT, AAG, ACA, ATA, AGA, CAA, TAA, GAA. All these words are recorded as k-mers found in the sequence. This statistic was introduced by Chan et al. [2], and is defined as follows:

$$D_2^n = \sum_{w \in A^k} X'_w Y'_w$$

Where $X'_w$ is word count of word $w$ in the sequence $X$ with the neigbourhood taken into account, and the same for $Y'_w$.

## Distance normalization

To transform the scores of one of the above statistics to an actual distance, we use the logarithmic representation of the geometric mean.

$$D_{ab} = \frac{S_{ab}}{\sqrt{S_{aa} \cdot S_{bb}}}$$

Where $D_{ab}$ is the actual distance between sequences $a$ and $b$, $S_{ab}$ is the score using one of the above statistics between sequences $a$ and $b$, and $S_{aa}$ and $S_{bb}$ are the self matching scores.

## 3 Python Implementation

Our Python program depends on a few external packages:

- NumPy
- Scikit-bio, which in turn depends on
  - Pandas
  - SciPy
  - matplotlib

NumPy is used for fast numerical computation, and scikit-bio provides us with a lot of the file format readers and writers, tree representations, and an implementation for neighbour-joining.

We've added some custom additions to scikit-bio: limited support for reading PHYLIp files, and a sliding window iterator for sequences.

### Distance calculation

To generate the k-mers of a sequence, we let a sliding window walk over the sequence. This iterator is passed to the built in Python `collections.Counter` class, which counts each returned element by a given iterator. It uses internally an optimized C implementation, so it's very fast.

## 4 Results

## A Source code

### $D_2$ statistic

```
def d2(seq1_iter, seq2_iter):
    """Compute D2 score for given sequences `seq1` and `seq2`.

    Parameters
    ----------
    seq1_iter : iterator yielding each word
        This parameter should be an iterator returning each k-length word in
        the first sequence.
```

```
    seq2_iter : string
        This parameter should be an iterator returning each k-length word in
        the second sequence.

    Notes
    -----
    D2 is defined as:

    .. math:: D_2 = \sum_{w \in A^k}X_w Y_w

    Where:

    :math:`X_w`
        The number of occurences of word `w` in sequence 1

    :math:`Y_w`
        The number of occurences of word `w` in sequence 2

    :math:`A^k`
        The set of all possible words of length k with alphabet A
    """

    word_count1 = Counter(seq1_iter)
    word_count2 = Counter(seq2_iter)

    self_matching1 = np.sum(np.array(word_count1.values())**2)
    self_matching2 = np.sum(np.array(word_count2.values())**2)

    logger.debug('%s', str(word_count1))
    logger.debug('%s', str(word_count2))

    return abs(math.log(
        sum(
            word_count1[word] * word_count2[word] for word in word_count1
        ) / math.sqrt(self_matching1 * self_matching2)
    ))
```


1. Wang L, Jiang T: **On the complexity of multiple sequence alignment**. *Journal of computational biology* 1994, **1**:337–348.

2. Chan CX, Bernard G, Poirion O, Hogan JM, Ragan MA: **Inferring phylogenies of evolving sequences without multiple sequence alignment**. *Scientific reports* 2014, **4**.

3. Katoh K, Standley DM: **MAFFT multiple sequence alignment software version 7: Improvements in performance and usability**. *Molecular biology and evolution* 2013, **30**:772–780.

4. Song K, Ren J, Reinert G, Deng M, Waterman MS, Sun F: **New developments of alignment-free sequence comparison: Measures, statistics and next-generation sequencing**. *Briefings in bioinformatics* 2013:bbt067.