Sumaclust: fast and exact clustering of sequences

metabarcoding.org/sumaclust

Introduction

With the development of next-generation sequencing, efficient tools are needed to handle millions of sequences in reasonable amounts of time. Sumaclust is a program developed by the LECA. Sumaclust aims to cluster sequences in a way that is fast and exact at the same time. This tool has been developed to be adapted to the type of data generated by DNA metabarcoding, i.e. entirely sequenced, short markers. Sumaclust clusters sequences using the same clustering algorithm as UCLUST and CD-HIT. This algorithm is mainly useful to detect the 'erroneous' sequences created during amplification and sequencing protocols, deriving from 'true' sequences. Currently, Sumaclust is available as a program that you can download and install on Unix-like machines.

Download and installation of Sumaclust

Download

Sumaclust can be downloaded from the metabarcoding.org GitLab. The archive of the latest tagged version can be downloaded on the GitLab wiki page:

https://git.metabarcoding.org/obitools/sumaclust/wikis/home

The versions downloaded this way are for Unix-like systems compatible with SIMD SSE2 instructions and POSIX threads. Pre-compiled versions of GCC for OS X can be found here, that might be helpful if you encounter problems compiling the programs. Send an email at celine.mercier@metabarcoding.org for other versions, or if you have any inquiries.

Installation

Untar the archive, go into the newly created directory and compile:

```
tar –zxvf sumaclust_v[x.x.xx].tar.gz
cd sumaclust_v[x.x.xx]
make
```

You can compile Sumaclust with `clang`, which deactivates `OpenMP`, with:

```
make CC=clang
```

Documentation

Sumaclust clusters sequences using the same clustering algorithm as UCLUST and CD-HIT. This algorithm is mainly useful to detect the "erroneous" sequences created during amplification and sequencing protocols, deriving from "true" sequences.

Using Sumaclust

Input

Input file must be in FASTA format.

Usage

```
sumaclust [-l|L|a|n|r|d|e|o|g|f] [-t threshold_value] [-s sorting_key] [-R maximum
_ratio] [-p number_of_threads] [-B file_name_for_BIOM-formatted_output] [-O file_n
ame_for_OTU_table-formatted_output] [-F file_name_for_FASTA-formatted_output] data
set
```

Argument: the sequence dataset to cluster.

For help :

```
sumaclust -h
```

Examples

```
sumaclust -t 0.97 my_dataset.fasta > clusters_of_seqs_with_similarity_>_97%.fasta
```

```
sumaclust -d -r -t 2 my_dataset.fasta > clusters_of_seqs_with_distance_<=_2_differ
ences.fasta
```

Options

```
-h : [H]elp - print the help
-l : Reference sequence length is the shortest.
-L : Reference sequence length is the largest.
-a : Reference sequence length is the alignment length (default).
-n : Score is normalized by reference sequence length (default).
-r : Raw score, not normalized.
-d : Score is expressed in distance (default : score is expressed in similarity).

-t ##.## : Score threshold for clustering. If the score is normalized and expresse
d in similarity (default),            it is an identity, e.g. 0.95 for an identity
 of 95%. If the score is normalized and expressed in distance, it is (1.0 - identi
ty), e.g. 0.05 for an identity of 95%. If the score is not normalized and expresse
d in similarity, it is the length of the Longest Common Subsequence. If the score
is not normalized and expressed in distance, it is (reference length - LCS length)
. Only sequences with a similarity above ##.## with the representative sequence of
 a cluster are assigned to that cluster. Default: 0.97.
-e : Exact option : A sequence is assigned to the cluster with the representative
sequence presenting the highest similarity score > threshold, as opposed to the de
fault 'fast' option where a sequence is assigned to the first cluster found with a
 representative sequence presenting a score > threshold.
-R ##    : Maximum ratio between the counts of two sequences so that the less abun
dant one can be considered             as a variant of the more abundant one. Defa
ult: 1.0.
-p ## : Multithreading with ## threads using openMP.
-s #### : Sorting by ####. Must be 'None' for no sorting, or a key in the fasta he
ader of each sequence, except for the count that can be computed (default : sortin
g by count).
-o : Sorting is in ascending order (default: descending).
-g : n's are replaced with a's (default: sequences with n's are discarded).
-B ###   : Output of the OTU table in BIOM format is activated, and written to fil
e ###.
-O ###   : Output of the OTU map (observation map) is activated, and written to fi
le ###.
-F ###   : Output in FASTA format is written to file ### instead of standard outpu
t.
-f : Output in FASTA format is deactivated.
```

Output

Sumaclust's default output is in fasta format. There are four fields added in the headers of all sequences. Those fields are of the form [key=value;]. The four keys are `cluster`, `cluster_score`, `cluster_center` and `cluster_weight` and their values correspond respectively to the identifier of the center of the sequence's cluster, the similarity score of the sequence with this center, a boolean indicating whether the sequence is the center of its cluster, and the total number of sequences in the cluster to which the sequence belongs.

Example where `seq_1` is a cluster center and `seq_2` is clustered with `seq_1`:

```
>seq_1 species=Heracleum maximum; count=3; cluster=seq_1; cluster_score=1.0; clust
er_center=True; cluster_weight=5; atcctattttccaaaaacaaacaaaggcccagaaggtgaaaaaag
>seq_2 species=Cnidium cnidiifolium; count=2; cluster=seq_1; cluster_score=0.95555
6; cluster_center=False; cluster_weight=5; atcctattttccaaaaacaacaaaggcccataaggtgaa
aaaag
```

There is a possibility to print the clusters in BIOM format with the `–B` option, and/or in OTU map (observation map) format with the `–O` option. The FASTA output can then be deactivated with the `–f` option. The FASTA output is written to the standard output by default, but can be written to a file using the `–F` option. In the following examples, the first one prints results in FASTA and BIOM formats, and the second one prints results in BIOM and OTU map formats:

```
sumaclust -B clusters_of_seqs_with_similarity_>_97%.biom my_dataset.fasta > cluste
rs_of_seqs_with_similarity_>_97%.fasta
```

```
sumaclust -F -B clusters_of_seqs_with_similarity_>_97%.biom -O clusters_of_seqs_wi
th_similarity_>_97%.txt my_dataset.fasta
```

How SUMACLUST works

Clustering algorithm

Sumaclust clusters sequences using the same clustering algorithm as UCLUST and CD-HIT. The problem is defined as follows:

Sumaclust browses through the dataset, in the order in which the sequences have been sorted with the -s option. By default, sequences are sorted by decreasing abundance, because this enables to identify 'true' and 'erroneous' sequences the best, as 'true' sequences tend to end up as cluster centers. The first sequence of the ordered list is considered the center of the first cluster. Each sequence, following the ordered list, is compared with the centers of the existing clusters, respecting the initial list's order. If the similarity of the query sequence with a center is above a chosen threshold, and their abundance ratio is below the maximum ratio chosen, the sequence is grouped in the cluster of this center. Otherwise, a new cluster is created with the query sequence as the center.

About the abundance ratio

An edge is created between a query sequence and a center sequence only if their abundance ratio, i.e. the query sequence’s count divided by the center sequence’s count, is below the maximum ratio chosen with the `–R` option. This can prevent sequences that are very abundant, and therefore likely true sequences, to be considered a variant of another true sequence that is only a little more abundant and very close to them.

Similarity computation

Similarity indice

A good way to evaluate the similarities between full-length sequences is to use indices based on the length of the Longest Common Subsequence (LCS), and in particular, a good similarity indice is the length of the LCS divided by the length of the shortest alignment representing this LCS, giving an identity percentage. This is the similarity indice used by Sumatra by default. Other similarity indices are available through the options.

Fast computation of the similarity

Lossless k-mer filter. Since we are usually interested in higly similar sequences, Sumatra uses similarity thresholds under which similarities are not reported. A lossless filtering step enables to only align couples of sequences that potentially have an identity greater than the chosen threshold. This filter is based on the number of overlapping k-mers that the sequences must share in order to have an identity at least equal to the threshold. With typical DNA metabarcoding datasets (a few millions sequences of 50-300 bp and threshold around 90-95% id), we empirically determined that the most efficient filtering was achieved with 4-mers and 5-mers.

Alignment within a diagonal band. Alignments are computed using a Needleman-Wunsch algorithm. In the scoring system used, matches are rewarded by one point, and mismatches and insertions/deletions are not penalised. The computation of the length of the LCS and the length of the alignment by the NWS algorithm has a quadratic complexity in time. It is responsible for most of the computation time. At high identity thresholds, the alignment computation can be done only in a diagonal band of the alignment matrix, gaining a considerable amount of time depending on the threshold.

Parallelization. There are two levels of parallelization implemented in Sumatra. Both the filtering and the alignments steps are optimized with the use of Simple Instruction Multiple Data instructions (SIMD). Since 4-mers enable to work easily with SIMD instructions, we implemented a 4-mer filter. Moreover, the program can be run on multiple threads.