
Clustering OTUs using U/VSEARCH

By Jonathan Palko and Thomas Coard

USEARCH



“Extreme high-throughput sequence analysis. Orders of magnitude faster than BLAST.” [4]

Usearch includes many utilities that are useful for analyzing read information.

The code is not open source and is mostly written by one person, Robert Edgar.

USEARCH Sequence Analysis Pipeline

Background: FASTQ Format

This is the output from the sequencing machines [5].

A FASTQ file normally uses four lines per sequence.

- Line 1 begins with a '@' character and is followed by a sequence identifier and an optional description (like a FASTA title line).
- Line 2 is the raw sequence letters.
- Line 3 begins with a '+' character and is optionally followed by the same sequence identifier (and any description) again.
- Line 4 encodes the quality values for the sequence in Line 2, and must contain the same number of symbols as letters in the sequence.

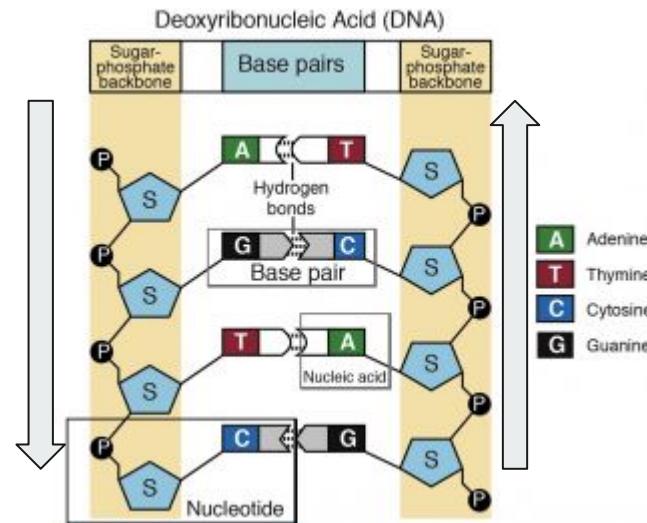
```
@M00967:43:00000000-A3JHG:1:1101:14069:1827 1:N:0:188
TACGGAGGATGCGAGCGTTATCCGGATTATTGGGTTAAAGGGTGCCTAGGCCTGCCAAGTCAGCGTA
+
3AA?ABBDBFFBEGGEGGGGAFFGGGGHHHCGGGGGGHFGHGGCFDEFGGGHGGGEGF1GGFGHHHHHGEG
```

Background: DNA Strands

Dna is made of two strands that are connected by nucleic acids.

Adenine and Thymine pair together.
Cytosine and Guanine pair together.

The strands are sequenced individually and are read in opposite directions.



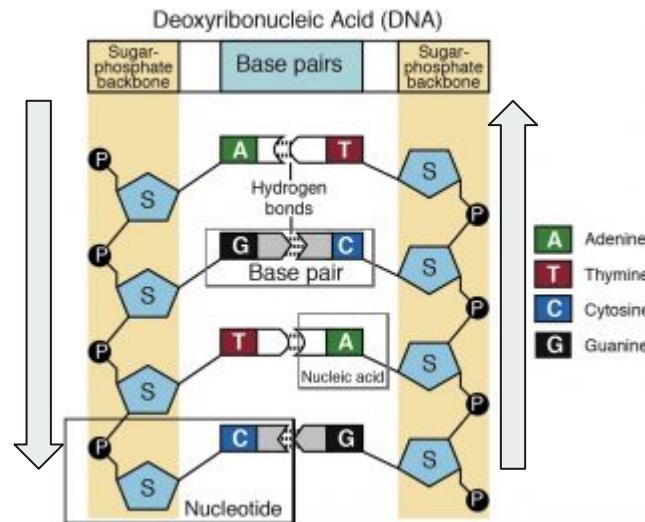
Merge Pairs

```
$usearch -fastq_mergepairs .. /fq/*_R1_*.fastq -relabel @ -fastqout merged.fq
```

This command will match all of the positive strand reads in the R1 files with the negative strand reads in the R2 files.

In the example on the right, AGTC would pair with GACT.

The output would be a new fastq that has the reads combined and averaged quality data for the read.



Discard Low Quality Reads

```
$usearch -fastq_filter merged.fq -fastq_maxee 1.0 -relabel Filt -fastaout filtered.fa
```

This command calculates the expected number of errors in the read and throws out any that are expected to have one or more error.

The output of this is a FASTA formatted file, which has a very similar format to FASTQ files, except FASTA files do not have read quality information.

Example:

```
>Filt4
TACGGAGGATGCGAGCGTTATCCGGATTATTGGGTTAAAGGGTGCAGGCCGAAGATCAAGTCAGCGGTAAAATTGA
GAGGCTAACCTCTCGAGCCGTTGAAACTGGTTTCTTGAGTGAGCGAGAAGTATGCGGAATGCGTGGTAGCGGTGA
AATGCATAGATATCACGCAGAACTCCGATTGCGAAGGCAGCATAACCGGCGCTCAACTGACGCTCATGCACGAAAGTGTGG
GTATCGAACAGG
```

Combine Unique Reads

```
$usearch -fastx_uniques filtered.fa -sizeout -relabel Uniq -fastaout uniques.fa
```

This command removes all duplicate reads and adds the total number of duplicate reads to the read header.

Example output sequence:

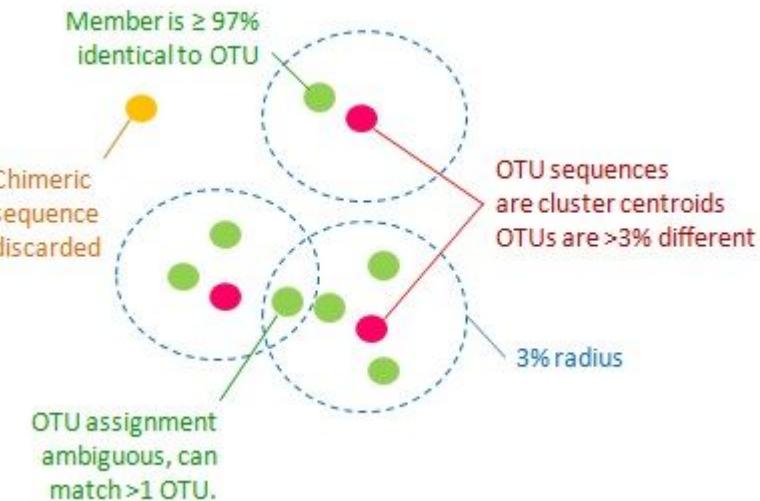
```
>Uniq5;size=4279;  
TACGGAGGATTCAAGCGTTATCCGGATTATTGGGTTAAAGGGTGCCTAGGCCTCGATAAGTTAGAGGTGAAATCCC  
GGGGCTCAACTCCGGCACTGCCTCTGATACTGTCGGGCTAGAGTTAGTTGCCTAGGCCTAGGCGGAATGTATGGTAGCGGTG  
AAATGCATAGAGATCATACAGAACACCGATTGCGAAGGCAGCTTACCAAACACTACGACTGACGTTGAGGCACGAAAGCGTG  
GGGAGCAAACAGG
```

Clustering OTUs

```
$usearch -cluster_otus uniques.fa -relabel Otu -otus otus.fa
```

This command clusters the reads into OTUs with ≥97% sequence similarity.

When choosing OTU sequences, USEARCH's clustering algorithm takes into account the abundance of the sequence.



OTU Table

```
$usearch -otutab merged.fq -otus otus.fa -strand plus -otutabout otutab.txt
```

This command generates an OTU table.

The Y axis is the OTUs generated and the X axis is the samples sequenced.

#OTU_ID	F3D0	F3D141	F3D142	F3D143	F3D144	F3D145	F3D146	F3D147	F3D148	F3D149	F3D150	F3D1	F3D2	F3D3	F3D5	F3D6	F3D7	F3D8	F3D9	Mock
Otu6	630	426	244	300	471	658	417	1637	1212	881	298	72	143	21	19	17	8	0	4	0
Otu2	343	254	157	103	220	311	199	864	510	482	170	242	2058	558	166	597	376	168	264	0
Otu29	68	20	6	13	14	10	22	11	9	42	30	69	74	8	41	45	9	29	42	0
Otu7	367	311	205	197	291	500	279	1181	780	707	297	117	433	230	146	357	251	181	267	1
Otu5	171	332	78	92	36	124	69	88	548	555	115	179	1233	412	190	250	233	281	425	0
Otu12	77	100	51	38	109	113	36	304	269	175	30	0	17	24	31	0	2	5	0	1
Otu19	104	15	6	2	7	8	44	24	50	41	22	257	563	19	88	73	27	51	99	0

USEARCH Exercises/Examples

USEARCH Exercise 1

- Use the syntax command to predict taxonomy for OTU sequences

```
$usearch -syntax .. /out/otus.fa -db .. /syntax/rdp_16s_v16.fa -strand plus -tabbedout .. /exout/otus_syntax.txt  
$usearch -syntax_summary .. /exout/otus_syntax.txt -rank p \  
-output .. /exout/syntax_summary.txt
```

Firmicutes	178	80.2	80.2
"Bacteroidetes"	15	6.8	86.9
(Unassigned)	9	4.1	91.0
"Actinobacteria"	8	3.6	94.6
"Proteobacteria"	7	3.2	97.7
Candidatus_Saccharibacteria	1	0.5	98.2
Cyanobacteria/Chloroplast	1	0.5	98.6
"Deinococcus-Thermus"	1	0.5	99.1
"Tenericutes"	1	0.5	99.5
"Verrucomicrobia"	1	0.5	100.0

USEARCH Exercise 2

```
$usearch -alpha_div .. /out/otutab.txt -metrics richness,shannon_2 -output .. /exout/alpha.txt
```

- Uses the alpha_div command to calculate richness and Shannon metrics
 - Uses base 2 (bits) for Shannon
- What is richness and Shannon diversity of mock sample?
 - Richness: 30
 - Shannon metric: 4.01

Sample	richness	shannon_2
F3D0	150.0	5.57
F3D141	144.0	5.03
F3D142	133.0	4.81
F3D143	127.0	5.02
F3D144	139.0	4.58

USEARCH Exercise 3

- Uses the otutab_norm command to normalize all the samples to 5000 reads and calculate the richness and Shannon diversity on the now normalized OTU table
- Besides rounding errors, this should not change our output because normalizing doesn't change whether an OTU is present (richness) or its frequency (Shannon).

```
$usearch -otutab_norm .. /out/otutab.txt -sample_size 5000 -output .. /exout/otutab_5k.txt  
$usearch -alpha_div .. /exout/otutab_5k.txt -metrics richness,shannon_2 -output .. /exout/alpha_5k.txt
```

USEARCH Exercise 4

- Uses the otus.uparseref file in misop/out_mock directory
 - Which OTU doesn't match a reference sequence from the mock community?
- Make a FASTA file with this non matching sequence
- Use the usearch_global command to find the same sequence in the OTU file for all samples combined (out/otus.fa)
 - Which is the corresponding OTU?
 - Why is this unexpected sequence found in reads for the mock sample?

Otu20 in out_mock/otus.fa is identical to Otu107 in out/otus.fa, likely due to cross-talk or a contaminant.

```
label=$(grep other .. /out_mock/otus.uparseref | cut -f1)

usearch -fastx_getseq .. /out_mock/otus.fa -label $label -fastaout .. /exout/$label.fa

usearch -usearch_global .. /exout/$label.fa -db .. /out/otus.fa -strand plus -id 0.97 \
-uc .. /exout/"$label"_hit.uc
```

USEARCH Exercise 5

- Runs the OTU sequence from Exercise 4 using NCBI BLAST
 - Which species (singular or plural) have BLAST hits with 100% identity?

We find 100% matches with many species in the *Pseudomonas* genus.
We found many more than when the tutorial was originally written.

The screenshot shows the NCBI BLAST search interface. In the 'Enter Query Sequence' section, a DNA sequence is pasted into the input field. In the 'Choose Search Set' section, the database is set to '16S ribosomal RNA sequences (Bacteria and Archaea)'.

Enter Query Sequence

Enter accession number(s), gi(s), or FASTA sequence(s) [Clear](#)

TACAGAGGGTCAACGCCCTAAATCGGAAATTACTGGGGCGTAAACCGCCGCCTA
GGTGCGTTCTTAAATGATGTGAAATCCG
CGGGCGTCACCCCGGAACTGCACTCAAACACTGACTGACTAGAGTATGGTA
GAGCGTCGIGAAATTCTCCTGCTACCGCGGTG

Query subrange? From To

Or, upload file Browse... No file selected. [?](#)

Job Title Nucleotide Sequence [?](#)

Enter a descriptive title for your BLAST search [?](#)

Align two or more sequences [?](#)

Choose Search Set

Database Standard databases (nr etc.) rRNA/ITS databases Genomic + transcript databases Betacoronavirus

Organism 16S ribosomal RNA sequences (Bacteria and Archaea) [Targeted Loci Project Information](#)

Exclude Models (XM/XP) Uncultured/environmental sample sequences

Limit to Sequences from type material

Entrez Query [Create custom database](#)

Program Selection

Optimize for Highly similar sequences (megablast) More dissimilar sequences (discontiguous megablast) Somewhat similar sequences (blastn)

Choose a BLAST algorithm [?](#)

BLAST Search database 16S ribosomal RNA sequences (Bacteria and Archaea) using Megablast (Optimize for highly similar sequences) Show results in a new window

Note: Parameter values that differ from the default are highlighted in yellow and marked with * sign

USEARCH Exercise 6

- Uses the syntax_summary command for phylum ranking
 - Which is the most common phylum in the OTUs?

Firmicutes	178	80.2	80.2
"Bacteroidetes"	15	6.8	86.9
(Unassigned)	9	4.1	91.0
"Actinobacteria"	8	3.6	94.6
"Proteobacteria"	7	3.2	97.7
Candidatus_Saccharibacteria	1	0.5	98.2
Cyanobacteria/Chloroplast	1	0.5	98.6
"Deinococcus-Thermus"	1	0.5	99.1
"Tenericutes"	1	0.5	99.5
"Verrucomicrobia"	1	0.5	100.0

VSEARCH

VSEARCH Overview

- Free, 64-bit open-source multithreaded sequence analysis tool
- Meant to be an alternative to USEARCH (which is proprietary)
- Functionality of VSEARCH is similar to USEARCH versions 7 and 8 in addition to unique functions
- This includes the following:
 - Searching based on exact or global alignment
 - Clustering by similarity (using length pre-sorting, abundance pre-sorting or user-defined order)
 - Chimera detection (referenced-based or de novo)
 - Dereplication (full length or prefix)
 - Pairwise alignment
 - Reverse complementation
 - Sorting
 - Subsampling
 - FASTQ file processing (format detection, filtering, read quality statistics, and merging of paired reads)

Unique VSEARCH Features

- VSEARCH has the following unique functions and features:
 - Shuffling
 - Re-replication
 - Masking of low-complexity sequences with the DUST algorithm (by Tatusov and Lipman)
 - The DUST algorithm focuses on masking of simple repeats and low-complexity nucleotide sequences
 - Full Code Algorithm:
<https://ftp.ncbi.nlm.nih.gov/pub/tatusov/dust/version1/src/dust.c>
 - Ability to choose among different similarity definitions
 - FASTQ file format conversion
- Uses pairwise alignments computed in parallel via vectorization over multiple threads



FAST PROCESSING

VSEARCH Searching Process

- VSEARCH's Searching Process Has Two Phases:
 - 1st Phase: Initial heuristic filtering based on shared words
 - Creates an index of 4^k possible distinct words, and stores the info about which database sequences each distinct word appears in (k is the number of consecutive nucleotides of a sequence, which is set to 8 by default)
 - 2nd Phase: Optimal alignment of the query with the most promising candidates
 - Considers the database sequences starting with the sequence that has the largest # of words in common with the query sequence
 - Comparison between the query sequence and database sequence is done by computing the optimal global alignment (Needleman & Wunsch algorithm)
 - This optimal alignment has HUGE memory requirements when aligning long sequences (>5000 base pairs)
 - If the two sequences have a product of their length greater than 20 million than an alternative method described by Hirschberg (1975) and Myers & Miller (1988) is used
 - This alternative methods is MUCH SLOWER but only uses a linear amount of memory
 - Computing the optimal pairwise alignment is each case is more computationally demanding but will provide more accurate results

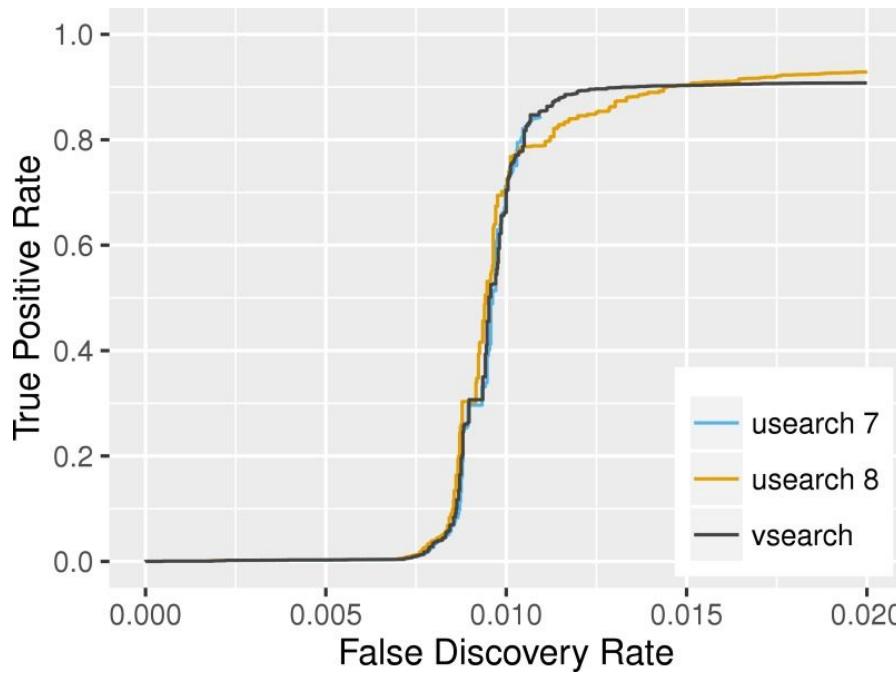
VSEARCH Clustering

- VSEARCH performs *de novo* clustering using greedy and heuristic centroid based algorithms with an adjustable sequence similarity threshold (which is specified with the *id* option)
 - The input sequences are used as a query in a search against an initially empty database of centroid sequences
 - The input sequences may be processed in any of the following orders:
 - Processed in user supplied order (*cluster_smallmem*)
 - Processed after being pre-sorted based on length (*cluster_fast*)
 - Processed after being pre-sorted based on abundance (*cluster_size*)
 - The input query sequence is clustered with the 1st centroid sequence found with similarity equal to or above the threshold (includes the sequence searching described previously)
 - **If no matches are found, the query sequence becomes the centroid of a new cluster and is added to the database**
 - **By default the query is clustered with the centroid presenting the highest sequence similarity (distance-based greedy clustering, DGC)**
 - If *sizeorder* option is ON then the centroid with the highest abundance is used (abundance-based greedy clustering)

USEARCH and VSEARCH Comparison

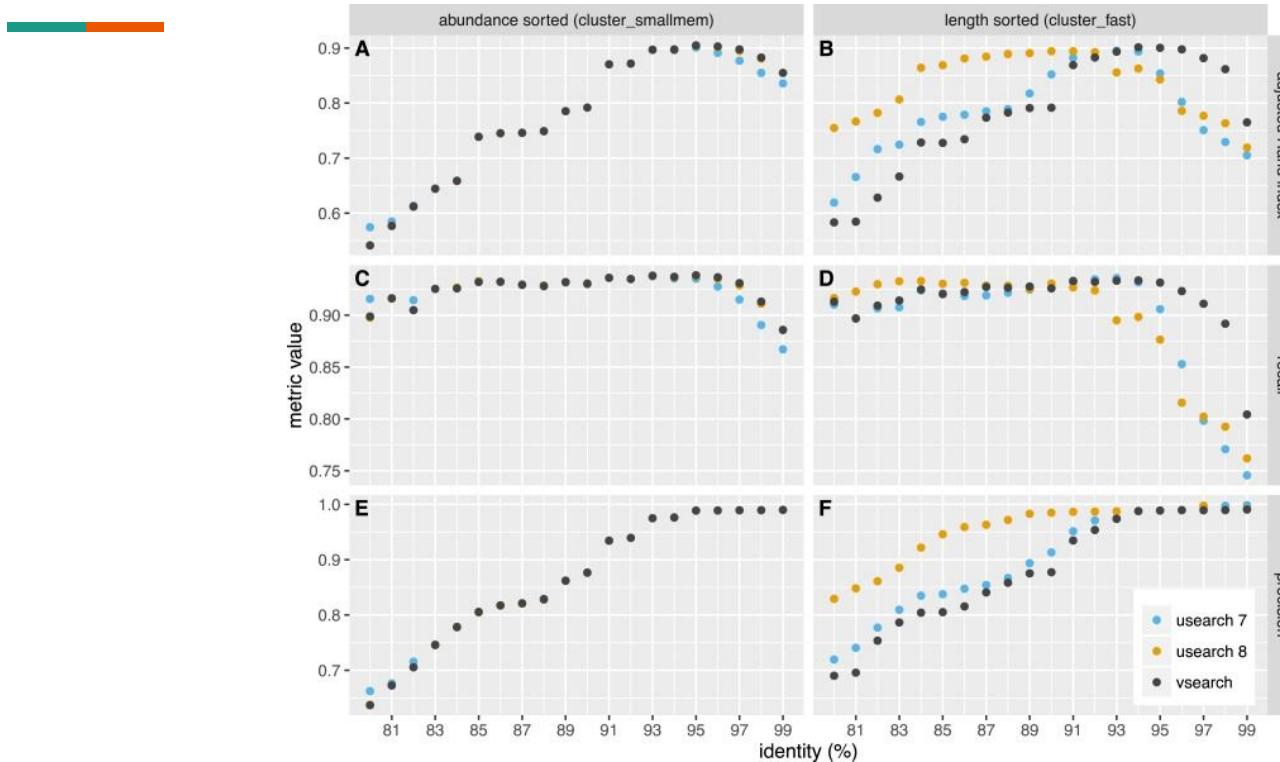
USEARCH	VSEARCH
Free 32-bit, max 4 Gb RAM version available	Free 64-bit, no maximum RAM version available
Only offers single-thread support	Supports multi-threading
Faster when performing chimera detection and clustering (see graphs)	More accurate when performing some chimera detection and clustering (see graphs)
USEARCH and VSEARCH have similar searching and subsampling speeds	More accurate when performing some searching and subsampling (see graphs)
About the same accuracy as VSEARCH when dealing with paired-ends read merging	About the same accuracy as USEARCH when dealing with paired-ends read merging, but with <u>faster</u> speed
Ineffective dereplication for shorter sequences	40-50% faster when performing de-replication

USEARCH and VSEARCH Search Comparison



Search accuracy on the RFAM v11 dataset^[2]

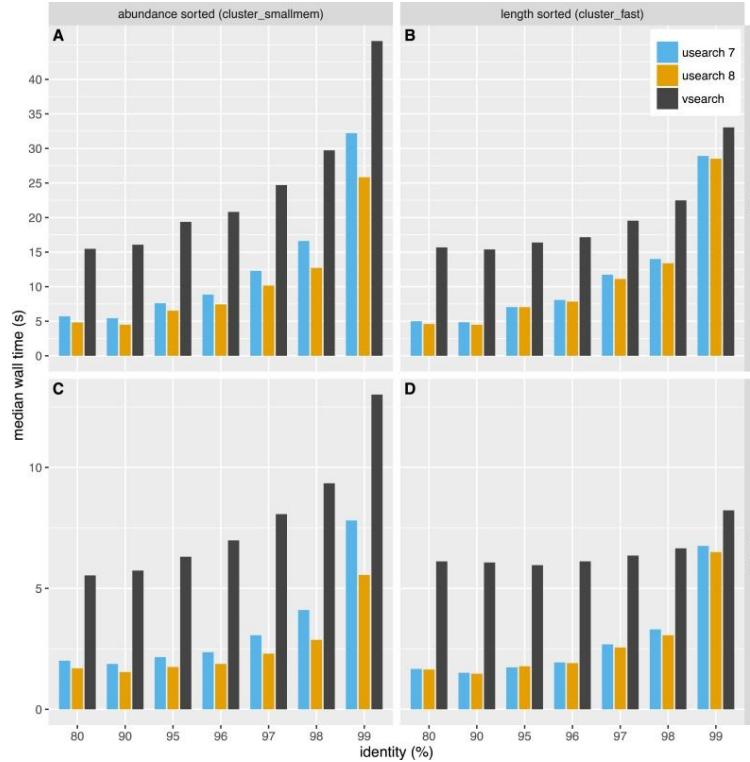
USEARCH and VSEARCH Clustering Comparison



Clustering accuracy using abundance sorting (A, C, E) and using length sorting (B, D, F)^[2]

USEARCH and VSEARCH Clustering Comparison

—



Clustering median time for even and uneven on the even dataset^[2]

References

- [1] https://www.researchgate.net/figure/Flow-chart-of-the-dust-detection-algorithm-The-number-associated-with-each-dust-test_fig2_216791182
- [2] <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5075697/>
- [3] <https://ftp.ncbi.nlm.nih.gov/pub/tatusov/dust/version1/src/>
- [4] <https://drive5.com/usearch/>
- [5] https://en.wikipedia.org/wiki/FASTQ_format
- [6] https://en.wikipedia.org/wiki/Needleman%20%93Wunsch_algorithm