CD-HIT User's Guide

Last updated: 2015/05/13 17:58

http://cd-hit.org Please visit http://cd-hit.org for most up-to-date documents

Program developed by Weizhong Li's lab at UCSD http://weizhong-lab.ucsd.edu and JCVI http://jcvi.org liwz@sdsc.edu

Introduction

CD-HIT is a widely used program for clustering biological sequences to reduce sequence redundancy and improve the performance of other sequence analyses.

CD-HIT was originally developed to cluster protein sequences to create reference databases with reduced redundancy (Li, et al., 2001) and was then extended to support clustering nucleotide sequences and comparing two datasets (Li and Godzik, 2006). The CD-HIT web server was implemented in 2009, which allows users to cluster or compare sequences without using command-line CD-HIT. The server provides interactive interface and additional visualization tools.

Currently, CD-HIT package has many programs: cd-hit, cd-hit-2d, cd-hit-est, cd-hit-est-2d, cd-hit-para, cd-hit-2d-para, psi-cd-hit, cd-hit-454, cd-hit-dup, cd-hit-lap, cd-hit-OTU etc. There are also many utility scripts, written in Perl, to help run and analyze CD-HIT jobs. Briefly:

```
* cd-hit        Cluster peptide sequences
* cd-hit-est    Cluster nucleotide sequences
* cd-hit-2d     Compare 2 peptide databases
* cd-hit-est-2d Compare 2 nucleotide databases
* psi-cd-hit    Cluster proteins at <40% cutoff
* cd-hit-lap    Identify overlapping reads
* cd-hit-dup    Identify duplicates from single or paired Illumina reads
* cd-hit-454    Identify duplicates from 454 reads
* cd-hit-otu    Cluster rRNA tags
* cd-hit Web server     Cluster user-uploaded data
* cd-hit-para   Cluster sequences in parallel on a computer cluster
* scripts       Parse results and so on
* h-cd-hit      Hierarchical clustering
```

Recently development of cd-hit, especially the multiple-threaded version introduced in 2012 (Fu et al), enables clustering of very large NGS datasets. For example, it only took cd-hit less than a day on a 32-core computer to cluster a few hundred million protein sequences from a metagenomics study.

Algorithm

[You can skip this session if you are eager to use the programs, no problem].

Algorithms for CD-HIT were described in these papers.

- 1. Weizhong Li, Lukasz Jaroszewski & Adam Godzik. Clustering of highly homologous sequences to reduce the size of large protein databases. Bioinformatics (2001) 17:282-283, PDF, Pubmed
- 2. Weizhong Li, Lukasz Jaroszewski & Adam Godzik. Tolerating some redundancy significantly speeds up clustering of large protein databases. Bioinformatics (2002) 18: 77-82, PDF, Pubmed
- 3. Weizhong Li & Adam Godzik. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. Bioinformatics (2006) 22:1658-1659 PDF, Pubmed
- 4. Limin Fu, Beifang Niu, Zhengwei Zhu, Sitao Wu and Weizhong Li. CD-HIT: accelerated for clustering the next generation sequencing data. Bioinformatics (2012) 28:3150-3152, doi: 10.1093/bioinformatics/bts565 PDF
- 5. Ying Huang, Beifang Niu, Ying Gao, Limin Fu and Weizhong Li. CD-HIT Suite: a web server for clustering and comparing biological sequences. Bioinformatics, (2010). 26:680 PDF Pubmed
- 6. Beifang Niu, Limin Fu, Shulei Sun and Weizhong Li, Artificial and natural duplicates in pyrosequencing reads of metagenomic data. BMC Bioinformatics, (2010) 11:187 PDF Pubmed
- 7. Weizhong Li, Limin Fu, Beifang Niu, Sitao Wu and John Wooley. Ultrafast clustering algorithms for metagenomic sequence analysis. Briefings in Bioinformatics, (2012) 13 (6): 656-668. doi: 10.1093/bib/bbs035 PDF

CD-HIT clustering algorithm

Clustering a sequence database usually requires all-by-all comparisons; therefore it is very time-consuming. Many methods use BLAST to compute the all vs. all similarities. It is very difficult for these methods to cluster large databases. While CD-HIT can avoid many pairwise sequence alignments with a short word based heuristics.

CD-HIT is a greedy incremental clustering approach. The basic CD-HIT algorithm sorts the input sequences from long to short, and processes them sequentially from the longest to the shortest. The first sequence is automatically classified as the first cluster representative sequence. Then each query sequence of the remaining sequences is compared to the representative sequences found before it, and is classified as redundant or representative based on whether it is similar to one of the existing representative sequences. In default manner (fast mode), a query is grouped into the first representative without comparing to other representatives. In accurate mode, a query is compared to all representatives and grouped to the most similar one.

Based on this greedy method, we established several integrated heuristics that make CD-HIT very efficient.

Index table: Most sequence alignment tools use short words or k-mer (k is the length) to speed up computation. For example, BLASTN uses 11-mer and BLASTP uses 3-mer by default. An index table, as opposite to a hash table used in most fast alignment methods, uses a unique index for every unique k-mer, therefore an index table is much faster than a hash table. In CD-HIT, we use k=2~5 for proteins and k=8~12 for DNAs, because the all the k-mers can be indexed in computer memory.

Short word filter: Two sequences of a certain identity must share at least a specific number of identical k-mers. It is possible to tell that the identity between two sequences is below a cutoff by counting common k-mers. The filter checks the common k-mers and rejects unnecessary alignments.

Short word statistics: It is the key for the high efficiency of a short word filter (Li, et al., 2002). Through statistical analysis of real alignments, we identified the distribution of common k-mer at different sequence lengths and identities, and applied the results to short word filter.

Banded alignment: The short word filter not only filters out unnecessary alignment, when alignment is needed, it also identifies a narrow band for banded dynamic programming alignment, which is much faster than regular dynamic programming.

Reduced alphabet (to be implemented): This is for protein clustering. In reduced alphabet, a group of exchangeable residues are reduced to a single residue (I/V/L==⇒I, S/T==⇒S, D/E==⇒D, K/R==⇒K, F/Y==⇒F), and then conservative mutations would appear as identities in sequence alignments. It improves the short word filter for clustering at low sequence identity below 50%.

Gapped word (to be implemented): Short word filter using gapped word allows mismatch within a word such as “ACE” vs “AME”, “ACFE” vs “AMYE”, and “AACTT” vs “AAGTT”, which can be written as “101”, “1001” and “11011”. At low identity cutoff, a gapped word is more efficient than an ungapped word for filtering.

Algorithm limitations

A limitation of short word filter is that it can not be used below certain clustering thresholds. For proteins:

- word size 5 is for thresholds 0.7 ~ 1.0
- word size 4 is for thresholds 0.6 ~ 0.7
- word size 3 is for thresholds 0.5 ~ 0.6
- word size 2 is for thresholds 0.4 ~ 0.5 (also see psi-cd-hit)

For DNAs:

- Word size 10-11 is for thresholds 0.95 ~ 1.0
- Word size 8,9 is for thresholds 0.90 ~ 0.95
- Word size 7 is for thresholds 0.88 ~ 0.9
- Word size 6 is for thresholds 0.85 ~ 0.88
- Word size 5 is for thresholds 0.80 ~ 0.85
- Word size 4 is for thresholds 0.75 ~ 0.8

Because of the algorithm, cd-hit may not be used for clustering proteins at <40% identity. Cd-hit-est cannot cluster very long sequences either (e.g. genome sized sequences). In such cases, please use PSI-CD-HIT, which will be introduced in following sections.

User's Guide

Installation

It can be copied under the GNU General Public License version 2 (GPLv2).

Most CD-HIT programs were written in C++. Installing CD-HIT package is very simple:

- download current CD-HIT at https://github.com/weizhongli/cdhit/releases, for example cd-hit-v4.6.2-2015-0511.tar.gz
- unpack the file with ” tar xvf cd-hit-v4.6.2-2015-0511.tar.gz --gunzip”

- change dir by “cd cd-hit-v4.6.2-2015-0511”
- compile the programs by “make” with multi-threading (default), or by “make openmp=no” without multi-threading (on old systems without OpenMP)
- cd cd-hit-auxtools
- compile cd-hit-auxtools by “make”

CD-HIT

CD-HIT clusters proteins into clusters that meet a user-defined similarity threshold, usually a sequence identity. Each cluster has one representative sequence. The input is a protein dataset in fasta format and the output are two files: a fasta file of representative sequences and a text file of list of clusters.

Basic command:

```
cd-hit -i nr -o nr100 -c 1.00 -n 5 -M 16000 –d 0 -T 8
cd-hit -i db -o db90 -c 0.9 -n 5 -M 16000 –d 0 -T 8,
```

where
`db` is the filename of input,
`db90` is output,
`-c 1.0`, means 100% identity, is the clustering threshold
`-c 0.9`, means 90% identity, is the clustering threshold
`-n 5` is the word size
`-d 0` use sequence name in fasta header till the first white space
`-M 16000`, to use 16GB RAM
`-T 8`, to use 8 threads

Choose of word size:

`-n 5` for thresholds 0.7 ~ 1.0
`-n 4` for thresholds 0.6 ~ 0.7
`-n 3` for thresholds 0.5 ~ 0.6
`-n 2` for thresholds 0.4 ~ 0.5

Complete options:

The most updated options are available from the command line version of the programs. Running the programs without any argument will print out the detailed options.

```
   -i   input filename in fasta format, required
   -o   output filename, required
   -c   sequence identity threshold, default 0.9
        this is the default cd-hit's "global sequence identity" calculated as:
        number of identical amino acids in alignment
        divided by the full length of the shorter sequence
   -G   use global sequence identity, default 1
        if set to 0, then use local sequence identity, calculated as :
        number of identical amino acids in alignment
        divided by the length of the alignment
        NOTE!!! don't use -G 0 unless you use alignment coverage controls
```

```
        see options -aL, -AL, -aS, -AS
   -b   band_width of alignment, default 20
   -M   memory limit (in MB) for the program, default 800; 0 for unlimitted;
   -T   number of threads, default 1; with 0, all CPUs will be used
   -n   word_length, default 5, see user's guide for choosing it
   -l   length of throw_away_sequences, default 10
   -t   tolerance for redundance, default 2
   -d   length of description in .clstr file, default 20
        if set to 0, it takes the fasta defline and stops at first space
   -s   length difference cutoff, default 0.0
        if set to 0.9, the shorter sequences need to be
        at least 90% length of the representative of the cluster
   -S   length difference cutoff in amino acid, default 999999
        if set to 60, the length difference between the shorter sequences
        and the representative of the cluster can not be bigger than 60
   -aL  alignment coverage for the longer sequence, default 0.0
        if set to 0.9, the alignment must covers 90% of the sequence
   -AL  alignment coverage control for the longer sequence, default 99999999
        if set to 60, and the length of the sequence is 400,
        then the alignment must be >= 340 (400-60) residues
   -aS  alignment coverage for the shorter sequence, default 0.0
        if set to 0.9, the alignment must covers 90% of the sequence
   -AS  alignment coverage control for the shorter sequence, default 99999999
        if set to 60, and the length of the sequence is 400,
        then the alignment must be >= 340 (400-60) residues
   -A   minimal alignment coverage control for the both sequences, default 0
        alignment must cover >= this value for both sequences
   -uL  maximum unmatched percentage for the longer sequence, default 1.0
        if set to 0.1, the unmatched region (excluding leading and tailing
gaps)
        must not be more than 10% of the sequence
   -uS  maximum unmatched percentage for the shorter sequence, default 1.0
        if set to 0.1, the unmatched region (excluding leading and tailing
gaps)
        must not be more than 10% of the sequence
   -U   maximum unmatched length, default 99999999
        if set to 10, the unmatched region (excluding leading and tailing
gaps)
        must not be more than 10 bases
   -B   1 or 0, default 0, by default, sequences are stored in RAM
        if set to 1, sequence are stored on hard drive
        it is recommended to use -B 1 for huge databases
   -p   1 or 0, default 0
        if set to 1, print alignment overlap in .clstr file
   -g   1 or 0, default 0
        by cd-hit's default algorithm, a sequence is clustered to the first
        cluster that meet the threshold (fast cluster). If set to 1, the
program
        will cluster it into the most similar cluster that meet the threshold
        (accurate but slow mode)
        but either 1 or 0 won't change the representatives of final clusters
```

```
   -bak write backup cluster file (1 or 0, default 0)
   -h   print this help
```

Alignment coverage control:

See the figure below, the -aL, -AL, -aS and -AS options can be used to specify the alignment coverage on both the representative sequence and other sequences. -s and -S can control the length difference between the representative sequence and other sequences.

$aL = R_a / R$
$AL = R - R_a$
$aS = S_a / S$
$AS = S - S_a$
$s = S_a / R_a$
$S = R / S$
$U = S_1 + S_2$
$uL = U / R$
$uS = U / S$

Output:

The output .clstr file looks like

```
>Cluster 0
0 2799aa, >PF04998.6|RPOC2_CHLRE/275-3073... *
>Cluster 1
0 2214aa, >PF06317.1|Q6Y625_9VIRU/1-2214... at 80%
1 2215aa, >PF06317.1|O09705_9VIRU/1-2215... at 84%
2 2217aa, >PF06317.1|Q6Y630_9VIRU/1-2217... *
3 2216aa, >PF06317.1|Q6GWS6_9VIRU/1-2216... at 84%
4 527aa, >PF06317.1|Q67E14_9VIRU/6-532... at 63%
>Cluster 2
0 2202aa, >PF06317.1|Q6UY61_9VIRU/8-2209... at 60%
1 2208aa, >PF06317.1|Q6IVU4_JUNIN/1-2208... *
2 2207aa, >PF06317.1|Q6IVU0_MACHU/1-2207... at 73%
3 2208aa, >PF06317.1|RRPO_TACV/1-2208... at 69%
```

where
a ”>” starts a new cluster
a “*” at the end means that this sequence is the representative of this cluster
a ”%” is the identity between this sequence and the representative

CD-HIT-2D

CD-HIT-2D compares 2 protein datasets (db1, db2). It identifies the sequences in db2 that are similar to db1 at a certain threshold. The input are two protein datasets (db1, db2) in fasta format and the output are two files: a fasta file of proteins in db2 that are not similar to db1 and a text file that lists similar sequences between db1 & db2.

Basic command:

```
cd-hit-2d -i db1 -i2 db2 -o db2novel -c 0.9 -n 5 -d 0 -M 16000 -T 8
```

where
`db1` & `db2` are inputs,
`db2novel` is output,
`0.9` means 90% identity, is the comparing threshold
`5` is the size of word

Please note that by default, cd-hit only lists matches where sequences in db2 are not longer than sequences in db1. You may use options -S2 or -s2 to overwrite this default. You can also swap db1 and db2:

```
cd-hit-2d -i db1 -i2 db2 -o db2novel -c 0.9 -n 5 -d 0 -M 16000 -T 8 -s2 0.9
cd-hit-2d -i db2 -i2 db1 -o db1novel -c 0.9 -n 5 -d 0 -M 16000 -T 8 (swap db1
and db2)
```

Choose of word size (same as cd-hit):

```
-n 5 for thresholds 0.7 ~ 1.0
-n 4 for thresholds 0.6 ~ 0.7
-n 3 for thresholds 0.5 ~ 0.6
-n 2 for thresholds 0.4 ~ 0.5
```

More options:

Options, -b, -M, -l, -d, -t, -s, -S, -B, -p, -aL, -AL, -aS, -AS, -g, -G, -T are same to CD-HIT, here are few more cd-hit-2d specific options:

```
-i2 input filename for db2 in fasta format, required
-s2 length difference cutoff for db1, default 1.0
    by default, seqs in db1 >= seqs in db2 in a same cluster
    if set to 0.9, seqs in db1 may just >= 90% seqs in db2
-S2 length difference cutoff, default 0
    by default, seqs in db1 >= seqs in db2 in a same cluster
    if set to 60, seqs in db2 may 60aa longer than seqs in db1
```

CD-HIT-EST

CD-HIT-EST clusters a nucleotide dataset into clusters that meet a user-defined similarity threshold, usually a sequence identity. The input is a DNA/RNA dataset in fasta format and the output are two files: a fasta file of representative sequences and a text file of list of clusters. Since eukaryotic genes usually have long introns, which cause long gaps, it is difficult to make full-length alignments for these genes. So, CD-HIT-EST is good for non-intron containing sequences like EST.

Basic command:

```
cd-hit-est -i est_human -o est_human95 -c 0.95 -n 10 -d 0 -M 16000 - T 8
```

Choose of word size:

```
-n 10, 11 for thresholds 0.95 ~ 1.0
-n 8,9    for thresholds 0.90 ~ 0.95
-n 7      for thresholds 0.88 ~ 0.9
-n 6      for thresholds 0.85 ~ 0.88
-n 5      for thresholds 0.80 ~ 0.85
-n 4      for thresholds 0.75 ~ 0.8
```

More options:

Options, -b, -M, -l, -d, -t, -s, -S, -B, -p, -aL, -AL, -aS, -AS, -g, -G, -T are same to CD-HIT, here are few more cd-hit-est specific options:

```
   -r   1 or 0, default 1, by default do both +/+ & +/- alignments
        if set to 0, only +/+ strand alignment
   -mask        masking letters (e.g. -mask NX, to mask out both 'N' and 'X')
   -match       matching score, default 2 (1 for T-U and N-N)
   -mismatch    mismatching score, default -2
   -gap gap opening score, default -6
   -gap-ext     gap extension score, default -1
```

CD-HIT-EST-2D

CD-HIT-EST-2D compares 2 nucleotide datasets (db1, db2). It identifies the sequences in db2 that are similar to db1 at a certain threshold. The input are two DNA/RNA datasets (db1, db2) in fasta format and the output are two files: a fasta file of sequences in db2 that are not similar to db1 and a text file that lists similar sequences between db1 & db2. For same reason as CD-HIT-EST, CD-HIT-EST-2D is good for non-intron containing sequences like EST.

Basic command:

```
cd-hit-est-2d -i mrna_human -i2 est_human -o est_human_novel -c 0.95 -n 10 -d
0 -M 16000 - T 8
```

Choose of word size and options are the same as CD-HIT-EST:

cd-hit-est-2d specificnoptions:

```
   -s2  length difference cutoff for db1, default 1.0
        by default, seqs in db1 >= seqs in db2 in a same cluster
        if set to 0.9, seqs in db1 may just >= 90% seqs in db2
   -S2  length difference cutoff, default 0
        by default, seqs in db1 >= seqs in db2 in a same cluster
        if set to 60, seqs in db2 may 60aa longer than seqs in db1
```

CD-HIT-454 clustering

We implemented a program called cd-hit-454 to identify duplicated 454 reads by reengineering cd-hit-est. Duplicates are either exactly identical or meet these criteria includes: (1) they start at the same position; (2) their lengths can be different, but shorter one must be fully aligned with the longer one (the seed); (3) they can only have 4% mismatches (insertion, deletion, and substitution);

and (4) only 1 base is allowed per insertion or deletion. Here, (3) and (4) can be adjusted by users. We allow mismatches in order to tolerate sequencing errors. To find duplicates in Illumina reads, please use cd-hit-dup (see later sections)

Basic command:

```
cd-hit-454 -i 454_reads -o 454_reads_95 -c 0.95 -n 10 -d 0 -M 16000 - T 8
```

Full list of options:

```
   -i   input filename in fasta format, required
   -o   output filename, required
   -c   sequence identity threshold, default 0.98
        this is a "global sequence identity" calculated as :
        number of identical amino acids in alignment
        divided by the full length of the shorter sequence + gaps
   -b   band_width of alignment, default 10
   -M   memory limit (in MB) for the program, default 800; 0 for unlimitted;
   -T   number of threads, default 1; with 0, all CPUs will be used
   -n   word_length, default 10, see user's guide for choosing it
   -aL  alignment coverage for the longer sequence, default 0.0
        if set to 0.9, the alignment must covers 90% of the sequence
   -AL  alignment coverage control for the longer sequence, default 99999999
        if set to 60, and the length of the sequence is 400,
        then the alignment must be >= 340 (400-60) residues
   -aS  alignment coverage for the shorter sequence, default 0.0
        if set to 0.9, the alignment must covers 90% of the sequence
   -AS  alignment coverage control for the shorter sequence, default 99999999
        if set to 60, and the length of the sequence is 400,
        then the alignment must be >= 340 (400-60) residues
   -B   1 or 0, default 0, by default, sequences are stored in RAM
        if set to 1, sequence are stored on hard drive
        it is recommended to use -B 1 for huge databases
   -g   1 or 0, default 0
        by cd-hit's default algorithm, a sequence is clustered to the first
        cluster that meet the threshold (fast cluster). If set to 1, the
program
        will cluster it into the most similar cluster that meet the threshold
        (accurate but slow mode)
        but either 1 or 0 won't change the representatives of final clusters
   -D   max size per indel, default 1
   -match       matching score, default 2
   -mismatch    mismatching score, default -1
   -gap gap opening score, default -3
   -gap-ext     gap extension score, default -1
   -bak write backup cluster file (1 or 0, default 0)
```

Multi-threaded programs

Multi-threaded cd-hit programs were implemented with OpenMP. Option ”-T n” will enable cd-hit to

run in parallel in a single multi-core computer. The default value of n is 1 (single thread). ”-T 0” will use all the cores in that computer. We have run cd-hit on 4-core, 8-core to 16-core computers and have observed a great speedup.

CD-HIT-PARA

[CD-HIT-PARA is no longer supported, since the multi-threaded cd-hit become available.]

CD-HIT-PARA is a script that runs cd-hit, cd-hit-est in a parallel mode. It splits the input database; runs cd-hit or cd-hit-est in parallel on a computer cluster; and finally merges the outputs into a single file. You can run it as you run cd-hit or cd-hit-est. The input is a protein or DNA/RNA dataset in fasta format and the output are two files: a fasta file of representative sequences and a text file of list of clusters.

There are two ways to run jobs on a cluster: by ssh to a remote computer and by queuing system (PBS and SGE are implemented). In any case, you should have a shared file system, the path to your working directory must be same on all the remote computers.

This script can also be used if you are clustering a very large database and your computer doesn't have enough RAM. In that case, all the divided jobs will still run on a single computer.

Implementation (see figure below)

1. divide input db into many small dbs in decreasing length
2. clusters the 1st db by cd-hit
3. run cd-hit-2d for other dbs against 1st db
4. repeat cd-hit and cd-hit-2d runs till done
5. Combine the results

Basic command:

```
cd-hit-para.pl -i nr90 -o nr60 -c 0.6 -n 4 --B hosts --S 64
```

where
`--B` hosts is a file with available hostnames
`--S` 64 is the number to split input db into, this number should be several times the number of hosts

More options:

```
--P program, "cd-hit" or "cd-hit-est", default "cd-hit"
--B filename of list of hosts,
    requred unless -Q or -L option is supplied
--L number of cpus on local computer, default 0
    when you are not running it over a cluster, you can use
    this option to divide a big clustering jobs into small
    pieces, I suggest you just use "--L 1" unless you have
    enough RAM for each cpu
--S Number of segments to split input DB into, default 64
--Q number of jobs to submit to queue queuing system, default 0
```

```
    by default, the program use ssh mode to submit remote jobs
--T type of queuing system, "PBS", "SGE" are supported, default PBS
--R restart file, used after a crash of run
```

CD-HIT-2D-PARA

[CD-HIT-2D-PARA is no longer supported, since the multi-threaded cd-hit become available.]

CD-HIT-2D-PARA is a script that runs cd-hit-2d, cd-hit-est-2d in a parallel mode. It splits the input databases; runs cd-hit-2d or cd-hit-est-2d in parallel on a computer cluster; and finally merges the outputs into a single file. You can run it as you run cd-hit-2d or cd-hit-est-2d. The input is a protein or DNA/RAN dataset in fasta format and the output are two files: a fasta file of representative sequences and a text file of list of clusters.

Basic command:

```
cd-hit-para.pl -i nr -i2 swissprot -o swissprot_vs_nr -c 0.6 -n 4 --Q 20 -T
"SGE" --S 2 --S2 20
```

where

```
    --P  program, "cd-hit-2d" or "cd-hit-est-2d",
         default "cd-hit-2d"
    --B  filename of list of hosts,
         requred unless -Q or -L option is supplied
    --L  number of cpus on local computer, default 0
         when you are not running it over a cluster, you can use
         this option to divide a big clustering jobs into small
         pieces, I suggest you just use "--L 1" unless you have
         enough RAM for each cpu
    --S  Number of segments to split 1st db into, default 2
    --S2 Number of segments to split 2nd db into, default 8
    --Q  number of jobs to submit to queue queuing system, default 0
         by default, the program use ssh mode to submit remote jobs
    --T  type of queuing system, "PBS", "SGE" are supported, default PBS
    --R  restart file, used after a crash of run
     -h  print this help
```

Incremental clustering

It is easy to make incremental update with cd-hit /cd-hit-2d. For example:

```
nr is the nr database of last month
month is the new sequences of nr of this month
```

In last month, you ran:

```
cd-hit -i nr -o nr90 -c 0.9 -n 5 -d 0 -M 16000 -T 16
```

This month, you can run incremental clustering

```
cd-hit-2d -i nr90 -i2 month -o month-new -c 0.9 -n 5 -d 0 -M 16000 -T 16
cd-hit -i month-new -o month90 -c 0.9 -n 5 -d 0 -M 16000 -T 16
cat month90 >> nr90
clstr_merge.pl nr90.clstr month-new.clstr > temp.clstr
cat temp.clstr month90.clstr > this_month_nr90.clstr
```

This approach is much faster than running from scratch. It also preserves stable cluster structure.

Hierarchically clustering

With multiple-step, iterated runs of CD-HIT, you perform a clustering in a neighbor-joining method, which generates a hierarchical structure. The third step use psi-cd-hit, please see psi-cd-hit section for details.

Commands:

```
cd-hit -i nr -o nr80 -c 0.8 -n 5 -d 0 -M 16000 -T 16
```

this generate nr80 and nr80.clstr

```
cd-hit -i nr80 -o nr60 -c 0.6 -n 4 -d 0 -M 16000 -T 16
```

this use nr80 to generate nr60 and nr60.clstr

```
psi-cd-hit.pl -i nr60 -o nr30 -c 0.3
```

this use nr60 to generate nr30 and nr30.clstr

```
clstr_rev.pl nr80.clstr nr60.clstr > nr80-60.clstr
```

nr60.clstr only lists sequences from nr80, script clstr_rev.pl add the original sequences from nr but not in nr80 into the output file nr80-60.clstr

```
clstr_rev.pl nr80-60.clstr nr30.clstr > nr80-60-30.clstr
```

nr30.clstr only lists sequences from nr60, script clstr_rev.pl add the original sequences into file nr80-60-30.clstr

This way is faster than one-step run from nr directly to nr30. It can also more accurate.

CD-HIT AuxTools

CD-HIT AuxTools is a set of auxiliary programs that can be used to assist the analysis of the next generation sequencing data. It currently includes programs for removing read duplicates, finding pairs of overlapping reads or joining pair-end reads etc.

cd-hit-dup

cd-hit-dup is a simple tool for removing duplicates from sequencing reads, with optional step to detect and remove chimeric reads. A number of options are provided to tune how the duplicates are removed. Running the program without arguments should print out the list of available options, as the following:

```
Options:
    -i        Input file;
    -i2       Second input file;
    -o        Output file;
    -d        Description length (default 0, truncate at the first whitespace
character)
    -u        Length of prefix to be used in the analysis (default 0, for
full/maximum length);
    -m        Match length (true/false, default true);
    -e        Maximum number/percent of mismatches allowed;
    -f        Filter out chimeric clusters (true/false, default false);
    -s        Minimum length of common sequence shared between a chimeric
read
              and each of its parents (default 30, minimum 20);
    -a        Abundance cutoff (default 1 without chimeric filtering, 2 with
chimeric filtering);
    -b        Abundance ratio between a parent read and a chimeric read
(default 1);
    -p        Dissimilarity control for chimeric filtering (default 1);
```

Option details

Common options

Here are the more detailed description of the options.

```
    -i        Input file;
```

Input file that must be in fasta or fastq format.

```
    -i2       Second input file;
```

cd-hit-dup can take a pair of files as inputs, assuming they contain sequences of pair-end reads. ”-i” can be used to specify the file for the first end; and ”-i2” can be used to specify the file for the second end.

When two files of pair-end reads are used as inputs, each pair of reads will be concatenated into a single one. And the following steps of duplicate and chimeric detection and removing.

```
    -o        Output file;
```

Output file which contains a list of reads without duplicates.

```
    -d        Description length (default 0, truncate at the first whitespace
character)
```

The length of description line that should be written to the output.

```
    -u        Length of prefix to be used in the analysis (default 0, for
full/maximum length);
```

For pair-end inputs, the program will take part (whole or prefix) of the first end and part (whole or prefix) of the second read, and join them together to form a single read to do the analysis. A positive value of this option specifies the length of the prefix to be taken from each read. If a read is shorter than this length, letter 'N's will be appended to the read to make up for the length. When this option is not used or is used with a non-positive value, the program will use the length of the longest read as the value of this option.

For single input analysis, only a positive value of this option will be effective. It also allows the program to use only the prefix up to the specified length of each read to do the analysis. In case that a read is shorter than this length, no 'N' is appended to the read since it is not necessary.

Options for duplicate detection

```
    -m        Match length (true/false, default true);
```

”-m” specifies whether the lengths of two reads should be exactly the same to be considered as duplicates.

```
    -e        Maximum number/percent of mismatches allowed;
```

Maximum number/percent of mismatches can be specified to control the similarity between two reads for duplicate and chimeric detection. For duplicate detection, any two reads with number of mismatches no greater than the specified value are considered to be duplicates. For chimeric detection, this option control how similar a read should be to either of its parents.

Options for chimeric filtering

```
    -f        Filter out chimeric clusters (true/false, default false);
```

This option specifies whether or not to carry out an additional step to filter out chimeric clusters.

```
    -s        Minimum length of common sequence shared between a chimeric
read
              and each of its parents (default 30, minimum 20);
```

A read or cluster representative is considered as a potential chimeric only if it shares at least the number of bases specified by this option with either of its parents. This option is effective only if the option is set to true for filtering chimeric clusters.

```
    -a        Abundance cutoff (default 1 without chimeric filtering, 2 with
chimeric filtering);
```

Each read is associated with an abundance number, which is the number of duplicates for the read. cd-hit-dup always assumes the input contains duplicates and perform the duplicate detection step. If no duplicate is found, the input is assumed to have duplicates remove in advance, and then, the program will try to obtain the abundance information from the descriptions of the reads, it interprets the number following "_abundance_" as the abundance number.

The abundance cutoff is mainly used for chimeric filtering to skip chimeric checking on reads with abundance below this cutoff.

```
    -b        Abundance ratio between a parent read and a chimeric read
(default 1);
```

This option specifies the abundance ratio between a parent read and a chimeric read. So for a read to be chimeric, either of its parents must have abundance at least as high as the ratio times the abundance of the chimeric read.

```
    -p        Dissimilarity control for chimeric filtering (default 1);
```

Internally dissimilarity is measured by percent of mismatches with ungapped alignments. By default the percentage cutoff is set to 0.01 (one percent). This option specifies a multiplier to this percentage cutoff. A higher value will increase the dissimilarity thresholds in chimeric filtering.

Output files

cd-hit-dup will output three files. Two of them are the same as the output files of CD-HIT: one (named exactly the same as the file name specified by the "-o" option) is the cluster (or duplicate) representatives, the other is the clustering file (xxx.clstr) relating each duplicate to its representative. The third file (xxx2.clstr) contains the chimeric clusters. In this file, the description for each chimeric cluster contains cluster ids of its parent clusters from the clustering file xxx.clstr.

Examples

Duplicate Detection

Remove duplicates using default parameters:

```
cd-hit-dup -i input.fa -o output
```

By default, only reads that are identical are considered as duplicates. If "-m" is set to false, duplicates will be allowed to have different length, but the longer ones must have a prefix that is identical to the shorter ones.

```
cd-hit-dup -i input.fa -o output -u 50
```

This only compare the first 50 bases of all sequences, considering that in Illumina reads, sequence quality are better at the beginning of the reads.

Remove duplicates with a few mismatches:

```
cd-hit-dup -i input.fa -o output -e 2
cd-hit-dup -i input.fa -o output -e 0.01
```

The former will allow each duplicate read to have up to 2 mismatches when aligned to its representative; and the later will allow up to one percent mismatches.

Remove duplicates from pair-end reads:

```
cd-hit-dup -i pair-end1.fa -i2 pair-end2.fa -o output
```

Each read from “pair-end1.fa” and “pair-end2.fa” will be joint to form a single read to detect duplicates. If they all are of the same length, the full length of each ends will be used in forming the single read; otherwise, the default value of option ”-u” will be used to determine how the single read is created.

```
cd-hit-dup -i pair-end1.fa -i2 pair-end2.fa -o output -u 50
```

This only consider the first 50 bases from both R1 and R2 reads.

Remove duplicates from pair-end reads with control on how the pair-ends are jointed:

```
cd-hit-dup -i pair-end1.fa -i2 pair-end2.fa -o output -u 100
```

With explicit ”-u” options, any reads shorter than 100 will be padded with 'N's, and the longer ones will be cut down to 100 base long. Then each pair of the 100 base long reads will be jointed to form a single 200 base long read.

Chimeric Filtering

cd-hit-dup offers a very efficient way to detect chimeric reads. The basic idea is to find two parent reads whose cross-over is sufficient similar to the chimeric read, while each single parent is sufficiently dissimilar to it.

Such dissimilarity is measured by the percent of mismatches for no-gapped alignments. For a given percentage “p” (from option ”-p”), a chimeric read must share at least “p” percent mismatches with any other single read, namely, it much be sufficiently dissimilar to any single read.

For more robust detection of chimeric reads, a background percentage “p_bg” is calculated as the mismatch percentage shared between the candidate chimeric read and the single read that is most similar to the candidate. If “p_bg” is greater than “1.5*p”, “1.5*p” will be used as “p_bg” instead.

For a read to be classified as chimeric read, there must exist two reads/parents such that, the leading part of the read is sufficiently similar to one parent, and the rest is sufficiently similar to the other parent, with at most “p+p_bg” percent of mismatches in each part. And the crossover between the two parents must share at most “p_bg” mismatches with the chimeric read.

Chimeric filtering with default parameters:

```
cd-hit-dup -i input.fa -o output -f true
```

Chimeric filtering with specified similarity level:

```
cd-hit-dup -i input.fa -o output -f true -p 1.5
```

Chimeric filtering with specified abundance difference:

```
cd-hit-dup -i input.fa -o output -f true -a 2
```

which means each parent of a chimeric read must be a least as twice abundant as the chimeric read.

Chimeric filtering will produce a cluster file named like “xxx2.clstr”, in which each cluster entry is a chimeric read/cluster. For example,

```
......
>Cluster 4 chimeric_parent1=2,chimeric_parent2=8
0   256nt, >FV9NWLF01CRIR3_abundance_23... *
>Cluster 5 chimeric_parent1=2,chimeric_parent2=0
0   250nt, >FV9NWLF01B4TBX_abundance_21... *
......
```

here “Cluster 5” contains a chimeric read “FV9NWLF01B4TBX”, whose parents are identified by cluster numbers “2” and “0” from the associated “xxx.clstr” file,

```
>Cluster 0
0   252nt, >FV9NWLF01ANLX2_abundance_2239... *
>Cluster 1
0   246nt, >FV9NWLF01C3KOB_abundance_1465... *
>Cluster 2
0   260nt, >FV9NWLF01AQOWA_abundance_1284... *
......
```

So the parent reads of the chimeric read “FV9NWLF01B4TBX” are “FV9NWLF01AQOWA” and “FV9NWLF01ANLX2”.

cd-hit-lap

cd-hit-lap is program for extracting pairs of overlapping reads by clustering based on tail-head overlaps (with perfect matching). The basic clustering strategy is the same as that in standard CD-HIT programs. In this program, each read is clustered as either a “representative” or a “redundant” read. For each “redundant” read, it must have a prefix that is identical a suffix of its representative read.

The options of this program can be obtained by running it it without any arguments:

```
[compute-0-0 cdhit-dup]$ ./cd-hit-lap
Options:
    -i        Input file;
    -o        Output file;
    -m        Minimum length of overlapping part (default 20);
    -p        Minimum percentage of overlapping part (default 0, any
percentage);
    -d        Description length (default 0, truncate at the first whitespace
character)
```

```
    -s        Random number seed for shuffling (default 0, no shuffling;
shuffled before sorting by length);
    -stdout   Standard output type (default "log", other options "rep",
"clstr");
```

The two options ”-m” and ”-p” can be used to control the minimum overlap that is required to classify them as overlapping reads. Each pair of overlapping reads must have overlap length no less than the threshold specified by ”-m”, and must also not be less than the length threshold computed from the ”-p” option.

Since the overlapping reads are searched using a greedy strategy, so different sortings of reads may lead to different result. So it is advisable to run the program multiple times with read shuffling by different random number seeds, and then collect and merge the results.

Sometimes it may be more convenient to pipe the results of this program as stdout directly to the stdin of other programs, to do this, the option ”-stdout” can be used to choose which type (“log” for program console information, “rep” for representative reads in FASTA or FASTQ format, “clstr” for the clustering output in CD-HIT format) of results to be writen to the stdout.

The output format of this program is the same as the standard CD-HIT. In the .clstr file, the alignment positions indicate how the reads are overlapped. For example,

```
>Cluster 0
0   75nt, >1_lane2_624... *
1   75nt, >1_lane2_7169... at 1:65:11:75/+/100.00%
2   75nt, >1_lane2_36713... at 69:1:1:69/-/100.00%
3   75nt, >1_lane2_141482... at 1:56:20:75/+/100.00%
```

The cluster member #0 in cluster #0 is the representative of the cluster, and it overlaps with each of the other members in the cluster. For cluster member #1, “1:65:11:75/+” tells that the first 65 bases of member #1 overlaps with the last 65 bases of member #0; “69:1:1:69/-” indicates that the last 69 bases of member #2 overlaps with the first 69 bases of member #0.

read-linker

read-linker is a very simple program to concatenate pair-end reads into single ones. It support the following options:

```
[compute-0-0 cdhit-dup]$ ./read-linker
Options:
    -1 file       Input file, first end;
    -2 file       Input file, second end;
    -o file       Output file;
    -l number     Minimum overlapping length (default 10);
    -e number     Maximum number of errors (mismatches, default 1);
```

Only the pairs of reads that share at least a minimum overlapping length with mismatched no more than the maximum number of errors, are jointed to form a single read.

PSI-CD-HIT clustering

The lowest threshold of CD-HIT is around 40%, in many applications, we need a much lower threshold, like 25%. Also CD-HIT-EST can not handle very long sequences (e.g. genomes, scaffolds).

PSI-CD-HIT clusters proteins at very low threshold, it also cluster long DNA sequences, through blastp, blastn and metablast. PSI-cd-hit is a Perl script, which runs similar incremental algorithm like CD-HIT, but using BLAST to calculate similarities. Below are the procedures of PSI-CD-HIT:

1. Sort sequences by decreasing length
2. First one is the first representative
3. Using 1st one blast all remaining sequences, pick up its neighbors that meet the clustering threshold
4. Repeat until done

Installation

please download legacy BLAST (not BLAST+) and install the executables in your $PATH. The programs required by psi-cd-hit.pl are blastall, megablast, blastpgp and formatdb.

Usage

Basic command:

```
psi-cd-hit.pl -i nr60 -o nr30 -c 0.3
```

More options:

```
  input/output:
    -i  in_dbname, required
    -o  out_dbname, required
    -l  length_of_throw_away_sequences, default 10

  thresholds:
    -c  clustering threshold (sequence identity), default 0.3
    -ce clustering threshold (blast expect), default -1,
        it means by default it doesn't use expect threshold,
        but with positive value, the program cluster seqs if similarities
        meet either identity threshold or expect threshold
    -G  (1/0) use global identity? default 1
        two sequences Long (i.e. representative) and Short (redunant) may
have multiple
        alignment fragments (i.e. HSPs), see:
        seq1  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Long sequence
                  ||||||||||||||||||            ////////////         i.e.
representative
                  ||||||||||||||||||           ////////////
```

```
sequence
                  |||||||HSP 1 ||||           ////HSP 2 ///
                  ||||||||||||||||||         ////////////
                  ||||||||||||||||||        ////////////
        seq2    xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Short sequence
                  <<  length 1    >>        <<   len 2 >>               i.e.
redundant
               <<<<<<<<<<< length of short sequence >>>>>>>>>>>>
sequence

                          total identical letters from all co-linear and
non-overlapping HSPs
        Glogal identity =
-------------------------------------------------------------------
                                       length of short sequence
        Local identity  = identity of the top high score HSP
        if you prefer to use -G 0, it is suggested that you also
        use -aS, -aL, such as -aS 0.8, to prevent very short matches.
    -aL alignment coverage for the longer sequence, default 0.0
        if set to 0.9, the alignment must covers 90% of the sequence
    -aS alignment coverage for the shorter sequence, default 0.0
        if set to 0.9, the alignment must covers 90% of the sequence
    -g  (1/0), default 0
        by cd-hit's default algorithm, a sequence is clustered to the first
        cluster that meet the threshold (fast cluster). If set to 1, the
program
        will cluster it into the most similar cluster that meet the threshold
        (accurate but slow mode)
        but either 1 or 0 won't change the representatives of final clusters
    -circle (1/0), default 0
        when set to 1, treat sequences as circular sequence.
        bacterial genomes, plasmids are circular, but their genome coordinate
maybe arbitary,
        the 2 HSPs below will be treated as non co-linear with -circle 0
        the 2 HSPs below will be treated as     co-linear with -circle 1
              -------------circle-----------
              |                            |
        seq1  xxxxxxxxxxxxxxxxxxxxxxxxxxxxx      genome / plasmid 1
               \\      ////////////
                \\    ////////////
                  HSP 2 -> ////HSP 1 ///   <-HSP 2
                           ////////////     \\
                          ////////////       \\
        seq2             xxxxxxxxxxxxxxxxxxxxxxxxxxxxx   genome / plasmid 2
                         |                            |
                         -----------circle-------------
   program:
     -prog (blastp, blastn, megablast, blastpgp), default blastp
     -p  profile search para, default
           "-j 3 -F F -e 0.001 -b 500 -v 500"
```

```
     -dprof database for building PSSM, default using input
         you can also use another database that is more comprehensive like
NR80
     -s  blast search para, default
           "-F F -e 0.000001 -b 100000 -v 100000"
     -bs (1/0) default 1
        pipe blast results from into parser instead of save in hard drive
(save time)

   compute:
     -exec (qsub, local) default local
          this program writes a shell script to run blast, this script is
          either performed locally by sh or remotely by qsub
          with qsub, you can use PBS, SGE etc
     -host number of hosts for qsub
     -core number of cpu cores per computer, default 1
     -shf a filename for add local settings into the job shell script
          for example, when you run PBS jobs, you can add quene name etc in
this
          file and this script will add them into the job shell script
e.g. your file may have followings
#PBS -v PATH
#PBS -l walltime=8:00:00
#PBS -q jobqueue

  job:
    -rs steps of save restart file and clustering output, default 5000
        everytime after process 5000 sequences, program write a
        restart file and current clustering information
    -restart restart file, readin a restart file
        if program crash, stoped, termitated, you can restart it by
        add a option "-restart sth.restart"
    -rf steps of re format blast database, default 200,000
        if program clustered 200,000 seqs, it remove them from seq
        pool, and re format blast db to save time
    -J  job, job_file, exe specific jobs like parse blast outonly
        DON'T use it, it is only used by this program itself
    -k (1/0) keep blast raw output file, default 0

    -P path to executables
```

Examples

Protein clustering

First, we use cd-hit to cluster the input down to 60% identity

```
cd-hit -i db    -o db_90 -c 0.9 -n 5 -g 1 -G 0 -aS 0.8  -d 0 -p 1 -T 16 -M 0
> db_90.log
```

```
cd-hit -i db_90 -o db_60 -c 0.6 -n 4 -g 1 -G 0 -aS 0.8  -d 0 -p 1 -T 16 -M 0
> db_60.log
```

Cluster on a single computer

```
  ./psi-cd-hit.pl -i db_60 -o db_30 -c 0.3 -ce 1e-6 -aS 0.8 -G 0 -g 1 -exec
local -core 16
  clstr_rev.pl db_90.clstr db_60.clstr > db90-60.clstr
  clstr_rev.pl db90-60.clstr db_30.clstr > db90-60-30.clstr
```

Here,

1. ce 1e-6 and -c 0.3 means cutoff at either 30% identity or 1e-6 e.value
2. G 0 means global identity
3. aS 0.8 means that alignment must cover 80% of shorter (redundant) sequence
4. g 1, slow but accurate mode, allowing sequences to be grouped to its most similar cluster
5. core 16, use 16 threads for blast search
6. clstr_rev.pl will combined all the cd-hit runs, see Hierarchically clustering

Cluster on a computer cluster (with qsub)

```
  ./psi-cd-hit.pl -i db_60 -o db_30 -c 0.3 -ce 1e-6 -aS 0.8 -G 0 -g 1 -exec
qsub -host 8 -core 8 -shf qsub_sh_template
  clstr_rev.pl db_90.clstr db_60.clstr > db90-60.clstr
  clstr_rev.pl db90-60.clstr db_30.clstr > db90-60-30.clstr
```

Restart:

```
  ./psi-cd-hit.pl -i db_60 -o db_30 -c 0.3 -ce 1e-6 -aS 0.8 -G 0 -g 1 -exec
local -core 16 -restart db_30.restart
  ./psi-cd-hit.pl -i db_60 -o db_30 -c 0.3 -ce 1e-6 -aS 0.8 -G 0 -g 1 -exec
qsub -host 8 -core 8 -shf qsub_sh_template -restart db_30.restart
```

In case of program crash, it restart from where it stops without re-running blast searches.

Clustering very long DNA sequences

```
  ./psi-cd-hit.pl -i db.fna -o db90.fna -c 0.9 -G 1 -g 1 -prog megablast -s
"-F F -e 0.000001 -b 100000 -v 100000" -exec local -core 32
  ./psi-cd-hit.pl -i db.fna -o db90.fna -c 0.9 -G 1 -g 1 -prog blastn -circle
1 -exec local -core 32
```

Here,

```
First example uses megablast
Second example uses blastn
-circle 1 means consider circular genome (see psi-cd-hit options above)
```

CD-HIT tools

cd-hit-div, cd-hit-div.pl

Both the executable binary program cd-hit-div and the perl script divide a FASTA file into pieces. The difference is that cd-hit-div sorts the sequences before dividing them while the perl script does not.

Commands:

```
cd-hit-div -i input -o output -div n
cd-hit-div.pl input output n
```

where “n” is the number of output files. The output files will be named as output-0, output-1 etc.

plot_len.pl

This is a script to print out distributions of clusters & sequences.

Commands:

```
plot_len.pl input.clstr \
1,2-4,5-9,10-19,20-49,50-99,100-299,500-99999 \
10-59,60-149,150-499,500-1999,2000-999999
```

where

```
2nd line are sizes of cluster
3rd line are lengths of sequences
```

It will print distribution of clusters and sequences :

```
Size    # seq   #clstr  10-59   60-149  150-499 500-1999 2000-up
1       266312  266312  36066   103737  103285  22727   497
2-4     208667  81131   1229    14680   44607   20006   609
5-9     156558  24198   118     2148    12026   9388    518
10-19   155387  11681   30      596     5024    5462    569
20-49   176815  6007    6       139     2212    3135    515
50-99   106955  1568    0       24      410     955     179
100-499 154209  896     0       3       124     597     172
500-up  43193   40      0       0       1       14      25
Total   1268096 391833  37449   121327  167689  62284   3084
```

clstr_sort_by.pl

This script sort clusters in .clstr file by length, size

Commands:

```
Clstr_sort_by.pl < input.clstr no > input_sort.clstr
```

Where, “no” means by size of the cluster

clstr_sort_prot_by.pl

This script sort sequences within clusters in .clstr file by length, name, etc.

Commands:

```
Clstr_sort_prot_by.pl input.clstr id > input_sort.clstr
```

Where, “no” means by id of sequences

clstr_merge.pl

It merges two or more .clstr files. The cluster orders need to be identical.

Commands:

```
cd-hit-2d -i db1 -i2 db2 -o db2new -c 0.9 -n 5
cd-hit-2d -i db1 -i2 db3 -o db3new -c 0.9 -n 5
clstr_merge.pl db2new.clstr db3new.clstr > db23new.clstr
```

clstr_merge_noorder.pl

It merges two or more .clstr files. The cluster orders do not have to be identical.

Commands:

```
cd-hit-2d -i db1 -i2 db2 -o db2new -c 0.9 -n 5
cd-hit-2d -i db1 -i2 db3 -o db3new -c 0.9 -n 5
clstr_merge_noorder.pl db2new.clstr db3new.clstr > db23new.clstr
```

clstr_ renumber.pl

It renumbers clusters and sequences within clusters in .clstr file after merge or other operations

Commands:

```
Clstr_renumber.pl input.clstr > input_ren.clstr
```

clstr_rev.pl

It combines a .clstr file with its parent .clstr file

Commands:

```
cd-hit -i nr -o nr90 -c 0.9 -n 5
cd-hit -i nr90 -o nr60 -c 0.6 -n 4
clstr_rev.pl nr90.clstr nr60.clstr > nr60_from90.clstr
```

```
psi-cd-hit -i nr60 -o nr30 -c 0.3
clstr_rev.pl nr60_from90.clstr nr30.clstr > nr30_from90.clstr
```

make_multi_seq.pl

This script reads the .clstr file, it generates a separate fasta file for each cluster over certain size and saves it in designated subdirectory. To run this script correctly, ”-d 0” option should be used in the cd-hit run and it is better to use ”-g 1” in the cd-hit run to get accurate clustering results. For example,

Commands:

```
cd-hit -i db -o dbout -c 0.6 -n 4 -d 0 -g 1
make_multi_seq.pl seq_db dbout.clstr multi-seq 20
```

will generate fasta files in “multi-seq” directory for clusters with more than 20 member sequences. Files will be named as “clusterN” where “N” is serial number of a cluster.

clstr2xml.pl

This script converts a cluster file or combines multiple cluster files from a hierarchical cd-hit run to xml format. The output is sorted by sequence length (default) or cluster size. The input cluster files must be in the order of being generated, that is, the cluster file with higher identity cutoff comes first.

Command:

```
clstr2xml.pl [-len|-size] input1.clstr [input2.clstr input3.clstr ...]
```

CD-HIT Web Server

The CD-HIT web server is available from http://cd-hit.org. All basic functions of CD-HIT are provided through tab-based interfaces in our web server. For CD-HIT and CD-HIT-EST, users can upload a FASTA file, select a desired sequence identity level and other parameters. CD-HIT-2D (CD-HIT-EST-2D) can compare two databases uploaded by users. H-CD-HIT and H-CD-HIT-EST in our server performs hierarchical clustering up to 3 steps.

The CD-HIT-454 web server is also available from http://cd-hit.org.

References

If you find cd-hit helpful to your research and study, please kindly cite the relevant references from the list below.

1. Weizhong Li, Lukasz Jaroszewski & Adam Godzik. Clustering of highly homologous sequences to reduce the size of large protein databases. Bioinformatics (2001) 17:282-283, PDF, Pubmed
2. Weizhong Li, Lukasz Jaroszewski & Adam Godzik. Tolerating some redundancy significantly speeds up clustering of large protein databases. Bioinformatics (2002) 18: 77-82, PDF, Pubmed
3. Weizhong Li & Adam Godzik. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. Bioinformatics (2006) 22:1658-1659, PDF, Pubmed
4. Ying Huang, Beifang Niu, Ying Gao, Limin Fu and Weizhong Li. CD-HIT Suite: a web server for clustering and comparing biological sequences. Bioinformatics, (2010). 26:680 PDF Pubmed
5. Beifang Niu, Limin Fu, Shulei Sun and Weizhong Li, Artificial and natural duplicates in pyrosequencing reads of metagenomic data. BMC Bioinformatics, (2010), 11:187 PDF Pubmed
6. Limin Fu, Beifang Niu, Zhengwei Zhu, Sitao Wu and Weizhong Li, CD-HIT: accelerated for clustering the next generation sequencing data. Bioinformatics, (2012), 28 (23): 3150-3152. doi: 10.1093/bioinformatics/bts565, PDF, Pubmed
7. Weizhong Li, Limin Fu, Beifang Niu, Sitao Wu and John Wooley. Ultrafast clustering algorithms for metagenomic sequence analysis. Briefings in Bioinformatics, (2012) 13 (6): 656-668. doi: 10.1093/bib/bbs035 PDF