# UCLUST

**Extreme high-speed sequence clustering, alignment and database search**

Robert C. Edgar
bob@drive5.com

http://www.drive5.com/uclust

Version 1.1.579
March 29th, 2010

Questions and requests for new features are welcome.

# Table of Contents

## Introduction

UCLUST is a high-performance clustering, alignment and search algorithm that is capable of handling millions of sequences.

Applications include:

Generating non-redundant protein and nucleotide sequence databases.

OTU-picking by clustering 16S/18S rRNA sequences.

Clustering reads from meta-genomic and environmental sequencing studies.

Taxonomy assignment of sequences to curated databases such as Greengenes, RDP and SILVA.

Phylogenetic analysis of very large sequence sets.



If you find this program useful, I would appreciate it if you would spread the word to your colleagues and add links from your web site (so that search engines will rank it higher). If you use UCLUST in published work, please cite my paper (submitted but not published at the time of writing) or this URL:

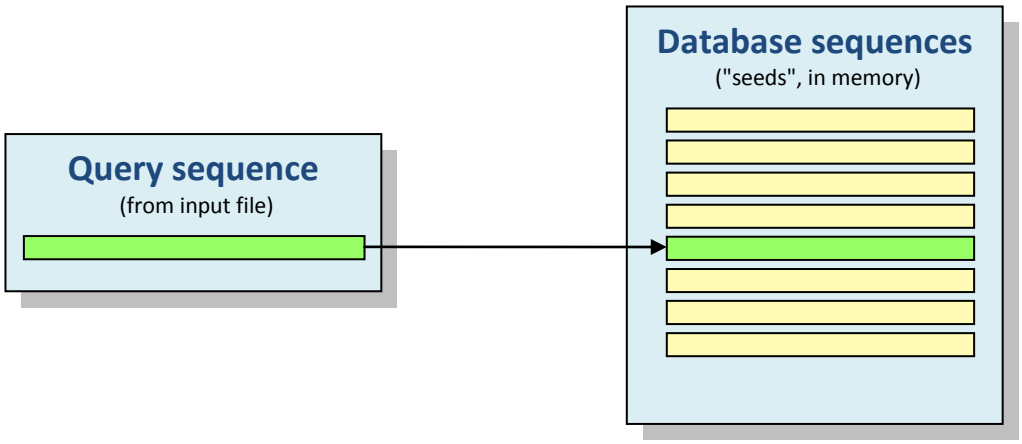
http://www.drive5.com/uclust

## How UCLUST works

UCLUST is a flexible program that can be used in many different ways, which can be confusing to new users. Here is an overview of how UCLUST works to help you get a picture of what's going on.

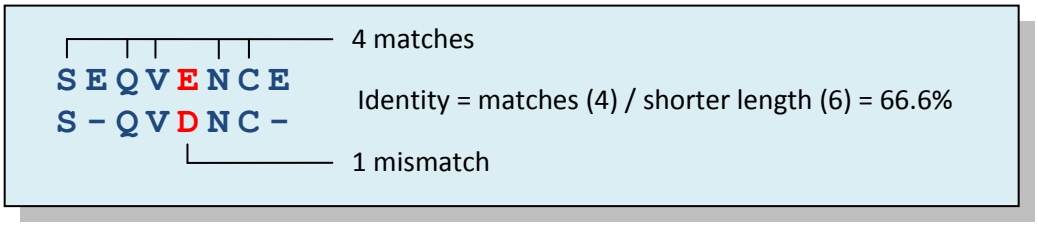
### Searching

The core step in the UCLUST algorithm is searching a database stored in memory.




A query sequence matches a database sequence if the identity is high enough. Identity is calculated from a global alignment, i.e. an alignment that includes all letters from both sequences. This differs from BLAST and most other database search programs, which search for local matches. A database sequence is sometimes called a *seed*, because a cluster of similar sequences can be grown from it.

The minimum identity is set by the --id option, e.g. --id 0.97 means that the global alignment must have at least 97% identity. Identity is computed as the number of matching (identical) letters divided by the length of the shorter sequence, as shown below.



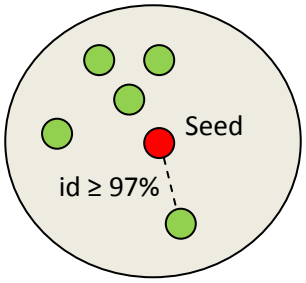

Other definitions of identity might be useful—if you would like additional options, let me know.

By default, UCLUST stops searching when it finds a match. Usually UCLUST finds the best match first, but this is not guaranteed. If it is important to find the best possible match (i.e., the database sequence with highest identity), then you can increase the --maxaccepts option, which defaults to 1. If you want all matches to be reported rather than the best match, then you can use the --allhits option, which will have no effect unless you also set maxaccepts > 1.

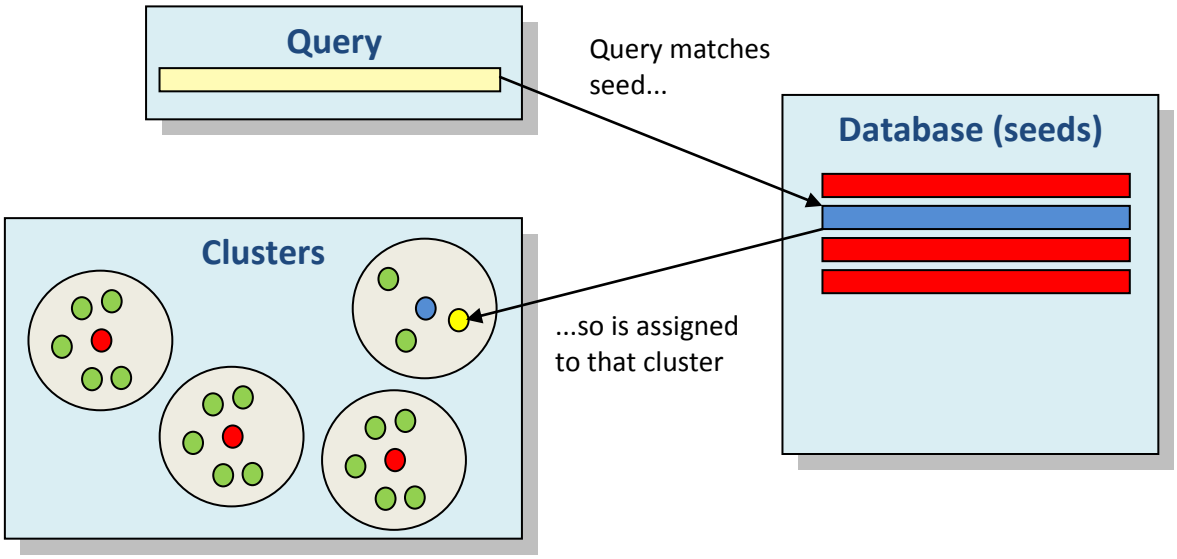
UCLUST also stops searching if it fails to find a match. By default, it gives up after eight failed attempts. Database sequences are tested in an order that correlates well (but not exactly) with decreasing identity. This means that the more sequences get tested, the less likely it is that a match will be found later, so giving up early doesn't miss a potential hit very often. You can set the maximum number to try using the --maxrejects option. With very high and very low identity thresholds, increasing maxrejects can significantly improve sensitivity. Here, a rule of thumb is that low identity is below 60% for amino acid sequences or 80% for nucleotides, high identity is 98% or more.

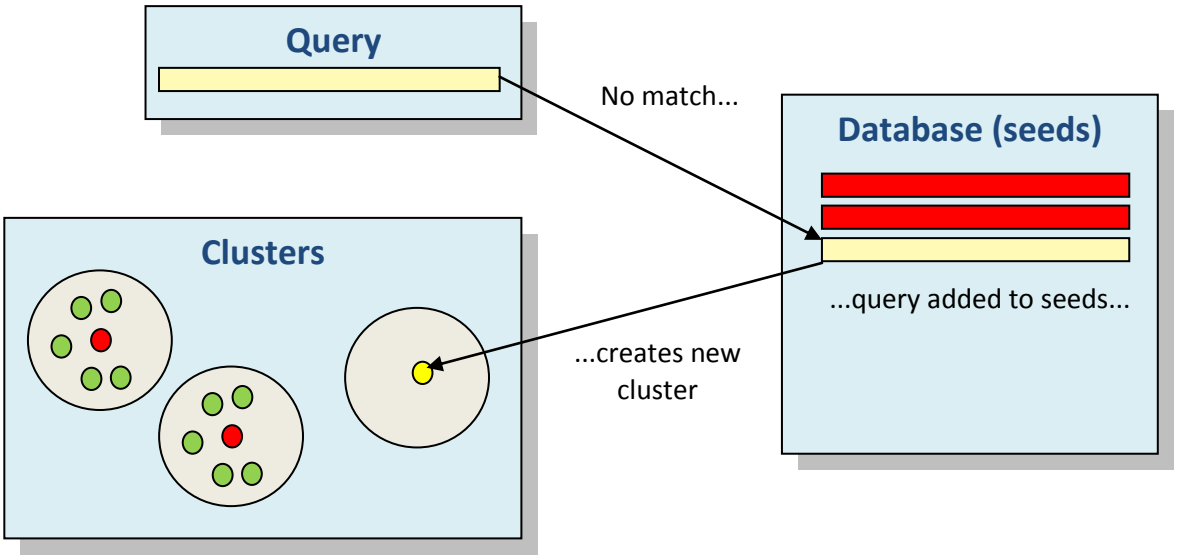
## Clustering

Each cluster is defined by a representative sequence called a *seed*. Each sequence in a cluster matches the seed according to the identity threshold, e.g. 97%.

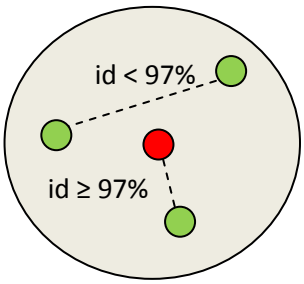



UCLUST performs *de novo* clustering by starting with an empty database in memory. Query sequences are processed in input order. If a match is found to a database sequence, then the query is assigned to that cluster (first figure below), otherwise the query becomes the seed of a new cluster (second figure below).






Only seeds need to be stored in memory (because other cluster members don't affect how new query sequences are processed). This is an advantage for large datasets because the amount of memory needed and the number of sequences to search are reduced. However, this design may not be ideal in some scenarios because it allows non-seed sequences in the same cluster to fall below the identity threshold. I plan to add other clustering methods to future versions of UCLUST: these will probably be slower for large datasets, but may be useful in some applications. Please let me know if there are new features you would like to have.




Pairs of non-seed sequences may have identities below the threshold.

## Libraries

You can initialize the seed database from an existing set of sequences (the *library*). Clustering is then pretty much the same thing as database search: UCLUST attempts to match each query sequence to a seed. The main difference is that by default, UCLUST stops when it finds the *first* hit, while most database search programs try to find *all* hits. The first hit is usually the best or close to the best possible hit, but this is not guaranteed (see the discussion of the --maxaccepts and --maxrejects options above).

## Extending the library

By default, queries that don't match the database become new seeds. If the database was initialized from a library, then conceptually this process extends the library. I say conceptually, because in practice UCLUST doesn't append new seeds to the FASTA file for the library while it is doing searches. You must

do this yourself by extracting 'S' (new seed) records from the output file, converting them to FASTA (the --uc2fasta option), and then appending them to the library, e.g. by using a command like:

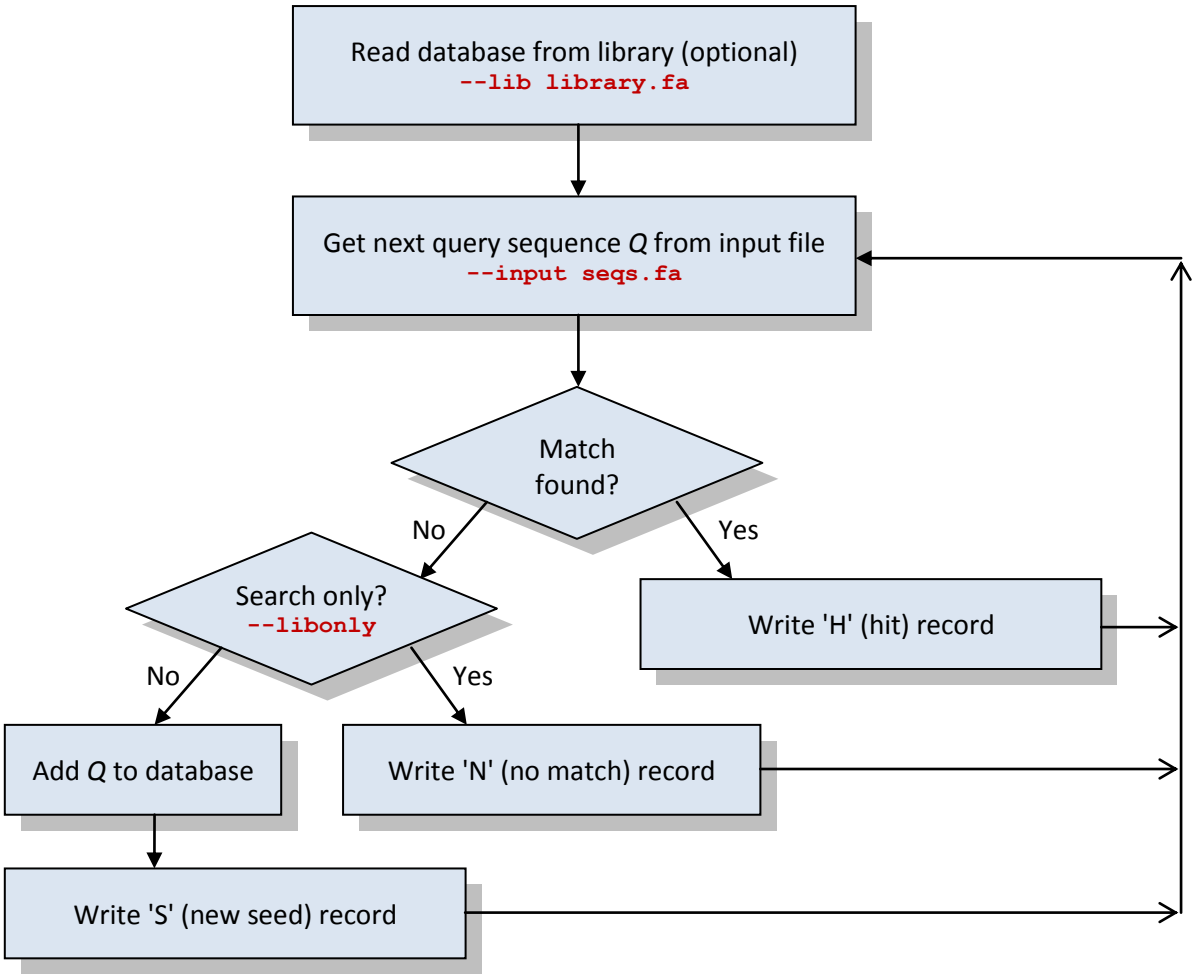
```
cat oldlib.fasta newseeds.fasta > newlib.fasta
```

### Search only

If you just want to search the library, you can disable the creation of new seeds by using the --libonly option.

### UCLUST flowchart

The typical flow of UCLUST is shown in the following chart. This may be modified depending on command-line options.



## Basic usage

In the following, uclust stands for the binary file name. Replace this with the appropriate file name on your system, e.g. uclust1.0.37_linuxi86_64.

### Clustering

UCLUST generates clusters containing similar sequences. A similarity threshold is specified, say 90% identity. Each cluster has a representative sequence (its *seed*); all sequences in a cluster are required to have identity ≥ *t* with the seed. Typical usage is:

```
uclust --sort seqs.fasta --output seqs_sorted.fasta
uclust --input seqs_sorted.fasta --uc results.uc --id 0.90
```

Here --id specifies the fractional identity threshold (90% in this example). Input is a FASTA file, output is generated in a .uc (UCLUST format) file.

### Database search

In database search mode, there are two input files: the database and the query set. Each sequence in the query set is compared with the database, which is searched for a match exceeding a specified identity threshold. The database (also called a library) file is specified using the --lib option, e.g.:

```
uclust --sort seqs.fasta --output seqs_sorted.fasta
uclust --input seqs_sorted.fasta --lib database.fasta --uc results.uc --id 0.90
```

By default, sequences that do not match the library become new seeds. Later sequences in the input file may be matched to these new seeds. To prevent the library from being extended in this way, use the --libonly option, as in the following example.

```
uclust --input seqs.fasta --lib database.fasta --uc results.uc --id 0.90 --libonly
```

### Multiple alignment

UCLUST can create a multiple alignment of each cluster. This requires three steps: 1. clustering, 2. conversion to FASTA (--uc2fasta), 3. inserting additional gaps (--staralign).

```
uclust --input seqs_sorted.fasta --uc results.uc --id 0.90
uclust --uc2fasta results.uc --input seqs_sorted.fasta --output results.fasta
uclust --staralign results.fasta --output aligned.fasta
```

To create a star alignment of all input sequences (the USTAR method), specify --id 0.0 to force all sequences into a single cluster. The first sequence in the input file is used as the 'center' of the star to which all other sequences are aligned. It is therefore recommended that you consider what the best center sequence would be and place it first. For example, short sequences or phylogenetic outliers would typically be bad choices.

## Input data

Input to UCLUST is generally in the form of FASTA files containing nucleotide or amino acid sequences. The library (database) is stored in memory. Input sequences are processed in the order they appear, allowing files of arbitrary size to be read sequentially with minimal use of memory. Input sequences should therefore be ordered so that the most appropriate seed sequence for a cluster is likely to be found before other members. For example, ordering by decreasing length is desirable when both complete and fragmented sequences are present, in which case full-length sequences are generally

preferred as seeds since a fragment may attract longer sequences that are dissimilar in terminal regions which do not align to the seed, as in the following example.

```
Seed:         THESEED
First hit:    THESEEDINSERTED
Second hit:   THESEEDTERMINAL
```

The two hits are both 100% identical to the seed in a pair-wise alignment (see later sections for a more detailed discussion of identity). However, the hits are extended with different terminal regions (red) and therefore have only about 50% identity to each other.

In other cases, long sequences may make poor seeds. For example, with some high-throughput sequencing technologies longer reads tend to have higher error rates, and in such cases sorting by decreasing read quality score may give better results.

By default, UCLUST checks that input sequences are sorted by decreasing length, unless --libonly is specified. This check can be disabled by specifying the --usersort option, which specifies that input sequences have been pre-sorted in a way that might not be decreasing length.

## Sorting input by length

Sorting by decreasing length is done in a separate step as follows:

```
uclust --sort input.fasta --output input_sorted.fasta
```

The current implementation of --sort loads all sequences into memory for faster speed, so requires that the available memory be at least as big as the input file. Larger sets can be sorted using a merge sort:

```
uclust --mergesort input.fasta --output input_sorted.fasta --split 500.0
```

The --split option (default 1000.0) specifies the number of megabytes to use for each partition of the input file. Typically, the maximum RAM needed for the sort will be a bit more than this, but in a worst-case scenario can be closer to 2x the --split value, so a conservative choice is to use about half the physically available memory. Smaller values tend to give slower speeds.

There is no need to sort sequences if --libonly is specified (because no new seeds are created).

## UCLUST file format

The native UCLUST format (.uc) is a tab-separated text file with one line for each sequence. The cluster number is always given. If there was a match to an existing seed, then the alignment to the seed and the identity computed from that alignment are also provided. A compressed representation of the alignment is used to save disk space. Records are appended to the output file as they are generated in order to minimize memory use, and sequences therefore appear in the same order as the input file.

Some example records:

| Type | Cluster | Size | %Id | Strand | Qlo | Tlo | Alignment | Query | Target |
|---|---|---|---|---|---|---|---|---|---|
| S | 0 | 292 | * | * | * | * | * | AH70_12410 | * |
| H | 0 | 292 | 99.7 | + | 0 | 0 | 292M | EN70_12566 | AH70_12410 |
| S | 1 | 292 | * | * | * | * | * | EX70_12567 | * |
| H | 1 | 292 | 98.2 | + | 0 | 0 | 292M | AH70_12410 | EX70_12567 |

Each record has ten fields, separated by tabs.

| | |
|---|---|
| Type | Record type |
| Cluster | Cluster number |
| Size | Sequence length or cluster size |
| %Id | Identity to the seed (as a percentage), or * if this is a seed. |
| Strand | + (plus strand), - (minus strand), or . (for amino acids). |
| Qlo | 0-based coordinate of alignment start in the query sequence. |
| Tlo | 0-based coordinate of alignment start in target (seed) sequence.<br>If minus strand, Tlo is relative to start of reverse-complemented target. |
| Alignment | Compressed representation of alignment to the seed (see below), or * if a seed. |
| Query | FASTA label of query sequence. |
| Target | FASTA label of target (seed / library / database) sequence, or * if a seed. |

Record types are:

- L Library seed (generated only if a match is found to this seed).
- S New seed.
- H Hit, also known as an accept; i.e. a successful match.
- D Library cluster.
- C New cluster.
- N Not matched (a sequence that didn't match library with --libonly specified).
- R Reject (generated only if --output_rejects is specified).

For records of type C and D, the Size field contains the cluster size, i.e. the number of sequences in the cluster including the seed, and %Id is the average identity of non-seed sequences to the seed. Otherwise, Size is the sequence length and %Id is the identity of the pair-wise alignment of this sequence to the seed. For Library clusters (D), records are only output if Size > 1, i.e. library sequences with no matches are not output. A library seed records (L) are output only if a hit is found to this seed. This saves writing a large number of records for library sequences that are not matched, but means that cluster numbers in the .uc file may not be consecutive (because UCLUST internally assigns a cluster number to every library seed, whether or not it is matched).

Rejections (R) are sequences that were aligned to a seed but found to have an identity below the threshold. Rejections are not output unless --output_rejects is specified. Using --output_rejects may increase the size of the .uc file significantly. Rejection records are mainly useful when trouble-shooting unexpected results.

The alignment is compressed using run-length encoding, as follows. Each column in the alignment is classified as M, D or I:

| Code | Name | Query sequence | Seed sequence |
|---|---|---|---|
| M | Match | Letter | Letter |
| D | Delete | Gap | Letter |
| I | Insert | Letter | Gap |

Here, "match" simply means a letter-letter column; the letters may or may not be identical. If there are *n* consecutive columns of type C, this is represented as *n*C. For example, 123M is 123 consecutive matches. As a special case, if *n*=1 then *n* is omitted. So for example, D5M2I3M represents an alignment of this form:

| | |
|---|---|
| Query sequence | `-XXXXXXXXXX` |
| Seed sequence | `XXXXXX--XXX` |
| Column type | `DMMMMMIIMMM` |

If a line in the output file starts with #, it is a comment and parser scripts should ignore it.

Records in the .uc file appear in the same order as the input sequences. You can sort the file using the standard Linux sort command, as follows:

```
sort -nk2 results.uc > results_sorted.uc
```

You can sort first by cluster number then by identity using:

```
sort -n -k2 -k4 results.uc > clusters.sorted.uc
```

UCLUST can also do the sort:

```
uclust --sortuc results.uc --output results_sorted.uc
```

However, the current implementation reads the entire file into memory, so may fail for very large sequence sets.

## FASTA format

UCLUST re-formats both labels and sequences when generating FASTA format output.

Labels look like this:

```
>43|99.7%|AH70_12410
```

Here, 43 is the cluster number and 99.7% is the identity to the seed. The identity will be shown as * for the seed:

```
>43|*|AH70_12200
```

If a .uc record has an alignment, then the query sequence is re-formatted to indicate its pair-wise alignment to the seed. Gaps indicate deletions relative to the seed, lower-case indicates insertions relative to the seed. Here is an example:

```
>43|99.7%|TheSeed
SEQVENCE
```

```
>43|96.0%|NonSeed
S-QLENnCE
```

This represents the following pair-wise alignment:

```
TheSeed    SEQVEN-CE
NonSeed    S-QLENNCE
```

You can convert UCLUST to FASTA format as follows:

```
uclust --uc2fasta results.uc --input seqs.fasta --output results.fasta [--types XYZ...]
```

Here, seqs.fasta must be the same input file used when generating results.uc. The --types option specifies which record types to convert, default is SH (seeds and hits). It is not valid to use L or D in --types because these refer to library sequences and the current implementation extracts sequences from the input set only. Using --types enables some convenient idioms, as in the following examples.

### *Create non-redundant database*

```
uclust --input seqs.fasta --uc results.uc --id 0.90
uclust --uc2fasta results.uc --types S --output nr.fasta
```

### *Assign to non-redundant database, cluster unmatched sequences and create new library*

```
uclust --input seqs.fasta --lib nr.fasta --uc results.uc --id 0.90
uclust --uc2fasta results.uc --types S --output newlib.fasta
```

## Multiple alignment FASTA format

Alignments generated by --uc2fasta are saved in a specialized FASTA format. You can convert to a more conventional multiple alignment format by using --staralign:

```
uclust --staralign results.fasta --output star.fasta
```

The results.fasta file must be sorted by cluster number.

Gaps are added so that each sequence in a given cluster has the same length. Letters that are aligned to the same position in the seed appear in the same column and are in upper case. Deletions relative to the seed are indicated by dashes. Insertions relative to the seed are indicated in lower-case, and should not be considered aligned to each other. The seed sequence is the last sequence is a special case that represents a consensus sequence. If the position is 100% conserved, i.e. if all letters in that column are identical, then an upper case letter is used. Otherwise, seed letters are lower-case.

## BLAST-like format

Alignments generated during clustering or database search can be saved in a human-readable BLAST-like format by using the --blastout option, e.g.:

```
uclust --input seqs.fasta --lib greengenes.fasta --libonly --blastout hits.blast
```

Since this format is rather verbose, the file size will be much larger than the corresponding .uc file. This format may be changed in future versions, so it is recommended that parsers use the FASTA output generated by --fastapairs instead.

## FASTA alignment format

Alignments generated during clustering or database search can be saved in FASTA format by using the --blastout option, e.g.:

```
uclust --input seqs.fasta --lib greengenes.fasta --libonly --fastapairs hits.fasta
```

This format is probably the most convenient for parsers that need to derive information from explicit alignments. Pairs are separated by blank lines, to make the file easier to inspect visually. The query sequence is first, the target (seed, database) sequence is second. If the input sequences are nucleotides, then a + or - is appended to the label of the target sequence to indicate the strand. If the strand is - (reverse strand match), then the target sequence is reverse-complemented.

## CD-HIT format

The CD-HIT .clstr format is supported for the benefit of code already written for that format. You can convert UCLUST format to and from .clstr as follows:

```
uclust --uc2clstr results.uc --output results.clstr

uclust --clstr2uc results.clstr --output results.uc
```

## "Exact" and "optimal" clustering

An "optimal" variant of the algorithm is specified by --optimal, which is equivalent to these options:

```
--maxaccepts 0 --maxrejects 0 --nowordcountreject
```

This guarantees that every seed will be aligned to the query, and that every sequence will therefore be assigned to the highest-identity seed that passes the identity threshold (*t*). All pairs of seeds are guaranteed to have identity < *t*. The number of seeds is guaranteed to be the minimum that can be discovered by greedy list removal, though it is possible that the number of clusters could be reduced by using a different set of seeds.

An "exact" variant of the algorithm is selected by --exact, which is equivalent to:

```
--maxaccepts 1 --maxrejects 0 --nowordcountreject
```

This guarantees that a match will be found if one exists, but not that the best match will be found.

The exact and optimal variants are guaranteed to find the minimum possible of clusters and both guarantee that all pairs of seeds have identities < *t*. Exact clustering will be faster, but may have lower average identity of non-seeds to seeds.

## Searching with reverse-complement (minus) strand

By default, UCLUST seeks nucleotide matches in the same orientation (i.e., plus strand only). You can enable both plus and minus strand matching by using --rev. In the current implementation, using --rev approximately doubles memory use but results in only small increases in execution time. Optimizations are possible that would avoid most of the increase in memory, but would be a fair amount of work to implement and so far do not appear to be worth the effort.

## Search parameter tuning

UCLUST offers a number of parameters for adjusting speed and sensitivity. The characteristics of datasets found in practice are highly variable, and it is therefore challenging to set universally appropriate defaults or to develop simple guidelines to assist users in setting the best parameter values for particular applications. With these considerations in mind, the following procedure is suggested for tuning parameters. (See also the section below *Evaluating fast alignment performance*).

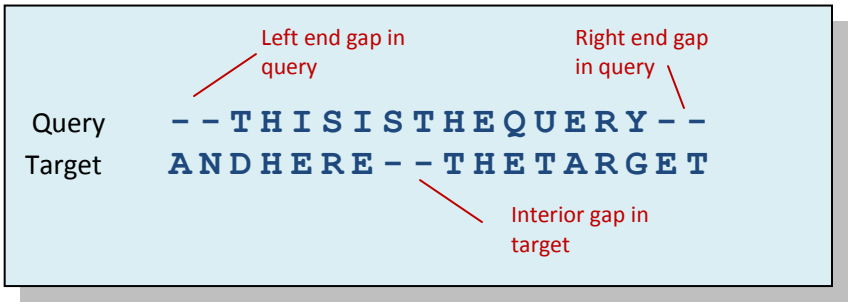
A dataset is first clustered at low identity (say, 50% for proteins or 80% for nucleotides) using default parameters, giving an initial set *S* of clusters. A subset *F* is then extracted from *S* for performance tuning analysis. The redundancy of *F* at higher identities, e.g. 90%, will typically be similar to *S*, meaning that the average cluster size will be similar. If *F* is small enough, the exact or optimal variants can be used as a reference against which faster but potentially less sensitive parameters can be compared. Redundancy is the most important factor in determining elapsed time (which scales roughly linearly in the number of sequences *N* and the number of clusters *M*) and memory use (which scales roughly linearly in *M*), unless *M* becomes very large. Time or memory use on *S* with a given set of parameters can therefore be estimated as $|S|/|F| \times$ (time or memory on *F*). If the exact variant is prohibitively expensive on *F*, an alternative is to use parameters designed for high sensitivity while retaining some speed heuristics, e.g.

--maxaccepts 1 --maxrejects 128 --step 0 --bump 0.

See also the --check_fast option, described next.

## Gap penalties

UCLUST supports a rich set of gap penalty options. Up to 12 separate penalties can be specified: all combinations of query / target, left / interior /end, and open / extend.




The following table gives the penalties that UCLUST uses by default.

| Penalty | Default |
|---|---|
| **Interior gap open** | 10.0 (nucleotide)<br>17.0 (amino acid) |
| **Terminal gap open** | 1.0 |
| **Interior gap extend** | 1 |
| **Terminal gap extend** | 0.5 |

Terminal gaps are penalized much less than interior gaps, which is typically appropriate when fragments are aligned to full-length sequences. These defaults can be changed using the --gapopen and --gapext options. The nucleotide defaults would be set using these options:

```
--gapopen 10.0I/1.0E --gapext 0.5
```

A numerical value for a penalty is optionally followed by one or more letters that specify particular types of gap. Here, "10.0I" means "Interior gap=10.0", and "1.0E" means "End gap=1.0". If no letters are given after the numerical value, then the penalty applies to all gaps. More than one letter can be specified, so for example "0.5IE" means "Interior and End gaps=0.5", which is the same as all gaps. Following are valid letters: I=Interior, E=End, L=Left, R=Right, Q=Query and T=Target. If more than one numerical value is specified, then they must be separated by a slash character '/'. White space is not allowed. If a star ('*') is used as the numerical value, then the gap is forbidden. Using * in an open penalty means that the gap will never be allowed, using * in an extension penalty means that gaps longer than one will be forbidden. So, for example, *LQ in --gapopen means "left end-gaps in the query are not allowed". A sign (plus or minus) is not allowed in the numerical value, which can be integer or floating-point (in which case a period '.' must be used for the decimal point). The --gapopen and --gapext options are interpreted first by setting the defaults, then by scanning the string left-to-right. Later values override previous values.

The final settings are written to the --log file, and I strongly recommend that you use this information to check that your options are correctly formatted. Here is another set of example options.

```
--gapopen 10.0QL/*QL/2.0TE/1.0QR --gapext 0.5I/0.1E
```

The resulting penalties appear as follows in the log file.

```
 10.00  Open penalty (query, internal)
     *  Open penalty (query, left end)
  1.00  Open penalty (query, right end)
 10.00  Open penalty (target, internal)
  2.00  Open penalty (target, left end)
  2.00  Open penalty (target, right end)
  0.50  Ext. penalty (query, internal)
  0.10  Ext. penalty (query, left end)
  0.10  Ext. penalty (query, right end)
  0.50  Ext. penalty (target, internal)
  0.10  Ext. penalty (target, left end)
  0.10  Ext. penalty (target, right end)
```

## Considerations when using non-standard gap penalties

The --gapopen and --gapext options do not always work well with the fast alignment heuristics that are enabled by default. In some cases, especially if some gap types are forbidden, then this can cause UCLUST to crash.

If possible, the best thing to do is to disable the heuristics by using --nofastalign. Then the gap penalties should work well. If you have very large datasets and heuristics are needed, then I recommend testing on small datasets and reviewing the --blastout file to make sure that the alignments look reasonable for your application.

A compromise that often works well is to disable HSPs by using --hsp 0. In typical applications, banding will still give improvements in speed without significantly degrading alignment accuracy or estimates of identity.

## Evaluating fast alignment performance

The --check_fast option performs an automated analysis of the --fastalign heuristics. With --check_fast, whenever a pair-wise alignment is constructed, UCLUST compares the alignment with and without fast heuristics (HSPs and banding). Results are written to the --log file. Here is an example.

```
Pair-wise alignment statistics
   1183407  Alignments
   1113873  Hits (94.1%)
    718023  Fast alignments same as slow (hits) (64.5%)
    774833  Fast alignments same as slow (all) (65.5%)
      4302  No HSPs found (0.4%)
       188  Alignments with HSPs not in slow (0.0%)
      3131  Rejected by low HSP id (0.3%), 0 are FPs (0.0%), 1 are FNs (0.0%)
      6319  Heuristic %id > 1% error vs. slow (0.5%)
         0  Heuristic false-positive hits (0.0%)
      4163  Heuristic false-negative hits (0.4%)
 2.65e+012  CPU time for slow alignments
 1.45e+011  CPU time for fast alignments
      18.3  Alignment time speedup by using heuristics
```

Here, 'slow' means without fast heuristics, i.e. using full dynamic programming as with --nofastalign. 1.1M alignments were made, and 94% of these were hits. This shows the effectiveness of the UCLUST index and filtering algorithm: almost all alignments confirmed putative hits. About 2/3 of alignments were the same with and without the --fastalign heuristics: 64.5% of alignments for hits, and 65.5% overall. So about 1/3 of alignments were sub-optimal in terms of maximizing the objective score. However, the statisitcs show that these sub-optimal alignments make little difference in estimating the query-target identity. There was just one false negative (FN) due to rejection of a target sequence based on the low identity of its HSP(s), and no false positives (FPs). In only 6.3k/1.1M cases (0.5%) was the identity estimated using a fast heuristic alignment more than 1% different from the identity computed from the full dynamic programming alignment. There were no false positive hits and only 4.2k/1.1M false negative hits, i.e. 0.4%. In most cases, this low "error" rate is well worth the 18x speed improvement achieved by using --fastalign. And in general, it is not certain that the full dynamic programming alignments are really better than the heuristic alignments, so it is not clear whether these are true biological "errors".

## Definition of identity

UCLUST computes identity from a global alignment as:

> (number of letter-letter columns containing identical letters) / (length of shorter sequence).

It is straightforward to modify UCLUST to support other definitions of identity—please let me know if another measure would be useful for you. Using the --blastout option is useful for visual review of alignments and identities.

## Computing all-vs-all pair-wise identities

You can compute all pair-wise identities for a set of input sequences as follows:

```
uclust --input seqs.fasta --lib seqs.fasta --usersort --allhits
  --libonly --maxaccepts 0 --maxrejects 0 --id 0 --uc allpairs.uc
```

# Command-line options

## Input options

| Option | Description |
|---|---|
| **--input filename** | Input file containing query sequences. FASTA format. By default, must be sorted by decreasing sequence length (see --usersort). |
| **--lib filename** | Library file. FASTA format. This is used to initialize the search database in memory. By default, the initial database is empty. |
| **--usersort** | By default, UCLUST requires that the input file is sorted by length, and will fail if it is not. Specify --usersort to indicate that file is sorted by some other criteria (or is not sorted at all but you think this is OK). For *de novo* clustering, input should be sorted so that a suitable seed sequence will appear before other members of the cluster. If the input includes both full-length sequences and fragments, then sorting by decreasing length is usually the best approach. |
| **--maxlen L** | Ignore query sequences that are longer than L. Default 10000. UCLUST is currently not designed to handle very long sequences. If you increase this value significantly, then UCLUST may fail due to lack of memory or for other reasons. Let me know if you have applications that need longer sequence lengths. |
| **--minlen L** | Ignore query sequences that are shorter than L. Default 16. |
| **--amino** | Specifies that input sequences use the 20-letter amino acid alphabet. By default, UCLUST 'guesses' this from the frequencies of AGCTU in the first few sequences. |
| **--nucleo** | Specifies that the input sequence use a nucleotide alphabet. By default, UCLUST 'guesses' this from the frequencies of AGCTU in the first few sequences. |

## Output options

| Option | Description |
|---|---|
| **--trunclabels** | Truncate FASTA labels at the first whitespace character. |
| **--allhits** | Write all hits to the .uc file. By default, only the best hit found is written. To get more than one hit, you must specify --allhits and set --maxaccepts to a value > 1. |

| Option | Description |
|---|---|
| | |
| **--[no]output_rejects** | Write reject records to the .uc file. By default, rejects are not written (--nooutputrejects). This is mainly useful for trouble-shooting unexpected results. |
| **--log filename** | Write a log file with miscellaneous information. I recommend that you try this and take a look at the output, you might find some of it helpful. |
| **--blastout filename** | Output file for human-readable alignments in a BLAST-like format. This format may be changed in future versions, so it is recommended that parsers use the FASTA output generated by --fastapairs instead. |
| **--fastapairs filename** | Output file for pair-wise alignments in FASTA format. Pairs are separated by blank lines, to make the file easier to inspect visually. The query sequence is first, the target (seed, database) sequence is second. If the input sequences are nucleotides, then a + or - is appended to the label of the target sequence to indicate the strand. If the strand is - (reverse strand match), then the target sequence is reverse-complemented. |
| **--rowlen n** | Row length for --blastout file. Default 64. |
| **--idchar c** | Character annotating identities in --blastout file. Default '\|'. |
| **--diffchar c** | Character annotating differences in --blastout file. Default blank . |
| **--[no]blast_termgaps** | [Don't] output terminal gaps to --blastout file. Default --noblast_termgaps. |

## Search options

| Option | Description |
|---|---|
| **--id f** | Minimum identity to accept a hit. Floating point number in range 0.0 to 1.0. Default 0.9. |
| **--maxaccepts n** | Keep searching until n hits have been found, then report the best. Default 1. Zero means infinity, i.e. don't stop however many matches have been found (but will still stop if the maximum number of rejects has occurred). Use --maxaccepts 0 --maxrejects 0 to force a search of the entire database with every query, this guarantees that the best hit will be found, if one exists. |
| **--maxrejects n** | Keep searching until n rejects have occurred, then report a failure to find a hit. Default 8. Zero means infinity, i.e. keep searching until all a hit is found or database sequences have been tested. |
| **--w n** | Word length for unique word index. Default is 8 for nucleotides, 5 for |

| Option | Description |
|---|---|
| | amino acids. It is not clear whether these defaults are good in all applications; further research is needed to understand this better. |
| **--[no]wordcountreject** | By default, --wordcountreject is enabled so that target sequences are rejected if they have too few unique words in common with the query sequence. The threshold is estimated using heuristics. This improves speed, but may also reduce sensitivity. Using --nowordcountreject disables word count rejection. |
| **--bump n** | By default, an optimization called "threshold bumping" is used to reduce the search space when many target sequences are found to pass the word count threshold. This may reduce sensitivity slightly, and may increase the probability that the top hit is not found, but often improves speed significantly when the database is large. Default is --bump 50. Use --bump 0 to disable bumping. |
| **--stepwords n** | By default, an optimization called "stepping" is used to speed up database searching. This is effective when the number of words in common between the query and target is expected to be large. Then it is expensive to check all words, and stepping selects a subset of words in the query. By default, --stepwords is 8. This means that the number of query words is chosen so that approximately 8 words are expected to be found in the target sequence. Use --stepwords 0 to disable stepping. As with bumping, stepping may reduce sensitivity and may reduce the probability that the best hit is found first. |
| **--rev** | By default, UCLUST searches only the plus strand for nucleotide sequences. If --rev is specified, then UCLUST will also search the reverse-complemented sequence. |
| **--libonly** | By default, if no hit is found, UCLUST will add the query sequence as a new seed. If --libonly is specified, this does not happen. Using --lib and --libonly is appropriate for database search applications. |
| **--self** | Used for searching a database against itself, e.g. to reduce redundancy. If the target and query labels are identical, the target is ignored. Typical use is uclust --input lib.fa --lib lib.fa --libonly --self --uc results.uc. |
| **--idprefix n** | Require that the first n letters of query and target are identical. Default zero. |
| **--exact** | Same as --maxrejects 0 --nowordfilter. Guarantees that a hit will be found if one exists. |
| **--optimal** | Same as --maxrejects 0 --maxaccepts 0 --nowordfilter. Similar to --exact but also guarantees that the best hit will be found. |

| Option | Description |
|---|---|
| | |

## Alignment options

| Option | Description |
|---|---|
| **--match s** | Match score for nucleotides. Default 2.0. |
| **--mismatch s** | Mismatch score for nucleotides. Default -1.0. |
| **--gapopen s** | Gap open penalty specification. Format is described elsewhere in this manual. |
| **--gapext s** | Gap extension penalty specification. Format is described elsewhere in this manual. |
| **--[no]fastalign** | Default is --fastalign. Specify --nofastalign to disable fast alignment heuristics (HSPs and banding). |
| **--hsp** | Minimum length for an HSP. Default 32. Specify zero to disable HSPs. |
| **--hspscore s** | Minimum score/column for an HSP. Default 1.0. |
| **--band n** | Radius of band for banded dynamic programming between HSPs. Default 16. Specify zero to disable banding. |
| **--check_fast** | Compare results using alignments with and without fast heuristics and generate a report in the --log file. |

## Miscellaneous options

| Option | Description |
|---|---|
| **--quiet** | Don't write progress messages to standard error. |
| **--version** | Write version to standard output and exit. |
| **--help** | Write command-line help to standard output and exit. |