Hammock - Hidden Markov Model based clustering of short peptide sequences

User's guide

version 1.2.0

Adam Krejci
Masaryk Memorial Cancer Institute
Brno, Czech Republic

March 16, 2018

News, info and downloads at:
https://github.com/krejciadam/hammock

We are happy to receive bug reports and questions. The preferred way of support is through the GitHub issues page :
https://github.com/krejciadam/hammock/issues
For direct email contact, please use:
krejciadam@gmail.com

Contents

1 Introduction

Hammock is a software tool for peptide sequence clustering based upon the usage of profile Hidden Markov Models. It is especially suitable for large datasets comprising of short sequences, e.g. data obtained by NGS deep sequencing of Phage Display libraries.

2 Citing Hammock

If you find Hammock useful, please cite this article:

Adam Krejci, Ted R. Hupp, Matej Lexa, Borivoj Vojtesek, and Petr Muller. Hammock: a hidden markov model-based peptide clustering algorithm to identify protein-interaction consensus motifs in large datasets. *Bioinformatics*, page btv522, sep 2015. doi: 10.1093/bioinformatics/btv522. URL `https://doi.org/10.1093/bioinformatics/btv522`

3 Prerequisites

Java, version at least 1.7.0 is required. Hammock also uses 3 external tools: Clustal Omega[6], Hmmer 3.0[2] and HHSuite[7]. Compiled versions of these tools are part of Hammock, but we strongly recommend to recompile them on your system, as compatibility and performance issues may rise otherwise. If you already have some of these tools installed or placed somewhere on your system, you may change the default path or command used by Hammock in the settings file (see 9.1 for details).

If HHSuite is located in other than the default Hammock/hhsuite-2.0.16 folder, you have to set up an environmental variable called `HHLIB` containing the path to HHSuite's `lib/hh` directory. This must be done before running Hammock. See the HHSuite manual for more details. Also note that the newest beta versions of HHSuite (3.x) are not compatible with Hammock.

4 How Hammock works

Hammock builds clusters gradually. Starting from single sequences, it first identifies initial clusters of very similar sequences using greedy or complete linkage clustering. After this step, several iterations of cluster extension and merging are performed.

4.1 Initial clustering

Initially, Hammock uses a simpler algorithm to generate groups of highly similar sequences - cluster cores. During this step, sequences are compared using a substitution matrix (controlled by `-m, --matrix` parameter, see 8.5), but full alignment is not performed. Instead, sequences are aligned as solid blocks, so that alignments with inner gaps are not considered. Moreover, sequences are only allowed to be shifted relative to each other by some maximal number of positions (specified by `-x, --max_shift` parameter, see 8.5).

There are two algorithms available for initial clustering. Full complete-linkage algorithm is more precise and guarantees optimal results for specified parameters. It is though more computationally demanding and by default, it is only used for datasets of up to 10 000 unique sequences. Greedy clustering is a much faster, yet less precise algorithm used for larger datasets. You can force Hammock to use one of these algorithms with `--use_clinkage` and `--use_greedy` parameters. See section 8.5 for details.

After this step, sequences are grouped into *cluster cores* - rather small groups of highly similar sequences.

4.2 Initial cluster extension

The number of cluster cores generated by initial clustering step is likely to be quite high. Therefore, Hammock only selects some of the largest clusters as true cluster cores (controlled by `-a, --part_threshold`, `-s, --size_threshold` or `-c, --count_threshold` parameter, see 8.8) to continue with. Before the next step, Hammock tries to extend these cores by merging them with some of the smaller clusters. HMM-HMM alignment of each cluster core to each remaining smaller cluster is performed and if similar enough (controlled by `-E, --initial_extension_threshold`, see 8.8), small clusters are merged into respective cluster cores.

4.3 Iterative cluster extension and merging

Small clusters that have not been selected as cluster cores, nor have been merged into cluster cores in the previous step are now "melted" back into single sequences. We will call this group of sequences the *sequence pool*.

Having cluster cores selected, Hammock constructs MSA and HMM for each of them. Using Hmmer's `hmmsearch` routine, it then searches the sequence pool for sequences similar to each core. Sequences similar enough (controlled by `-n, --assign_thresholds` parameter, see 8.8) are removed from the sequence pool and inserted into appropriate cluster (the one they are most similar to). We call this step the *extension step*.

During the extension step, some sequences may have been found to be similar to more then one cluster (the meaning of "similar" here is controlled by `-v, --overlap_thresholds` parameter, see 8.8), which means that the set of sequences similar to one cluster may have nonempty overlap with the set of sequences similar to another. This indicates that such two clusters are somewhat similar to each other themselves. If they are similar enough, Hammock will merge them into one cluster. The decision whether clusters are similar enough to be merged or not is made upon the result from HHSuite's `hhalign` routine (controlled by `-r, --merge_thresholds` parameter, see 8.8). Things may get more complicated when more than 2 clusters have their sets of similar sequences overlapping. Therefore, a special cluster-clustering scheme is used in this step, which we call the *merging step*.

Hammock tries to grow clusters from small groups of very similar sequences to bigger groups with less similarity, sharing just some (potentially short) strong motif. To achieve that, extension step and merging step are repeated iteratively several times (3 by default) while

similarity thresholds are gradually relaxed. This leads to progressive cluster growth, as the motif emerges.

Apart from thresholds, there is another system to assure that resulting clusters will not become too diverse. To represent cluster with HMMs, *match states* must be defined. A match state is virtually a column in a cluster's MSA which does not have too many gaps in it. In Hammock, a match state is any MSA column between the leftmost and the rightmost conserved position (This is not true if inner gaps are allowed. In that case, an inner MSA column containing gaps may lie between two match states). You can control which columns will be marked as conserved positions by `-y, --max_gap_proportion` and `-k, --min_ic` parameters and limit the minimal number of such positions by `-h, --min_conserved_positions` parameter. Match states are then defined accordingly. See section 8.8 for details.

5 Files and formats

5.1 Input files

In `full`, `clinkage` and `greedy` modes, Hammock uses single sequences as input. In `cluster` mode, the input consists of entire clusters.

5.1.1 Sequence input

There are 2 supported file formats for sequence input. The .fasta format and the .tsv (.csv) table format. Both these formats support the concept of sequence labels.

The sequence label concept In Hammock, the most basic piece of data is a unique sequence. If one unique sequence occurs in more than one copy in the original dataset, the information about the number of copies is preserved. To support more structured form of data, Hammock allows the user to define sequence labels.

A label marks a group of sequences forming some sub-dataset in the original dataset. An example of such sub-datasets is grouping the sequences from Phage display experiment according to the round of selection they were sequenced after. One unique sequence may be preset in different counts within groups with different labels.

If no label is specified, all sequences are labeled with default label `no_label`.

Labels must not contain pipe (`|`) and whitespace special characters.

.fasta format Sequences in classic .fasta format where each sequence is saved as a header line starting with `>` followed by one line containing the actual peptide sequence. The header line typically contains unique sequence id, but Hammock does not require the sequence id to be unique. Actually it does not require the sequence id to be even present - the only limit on the header line is that it must start with "`>`" character.

One unique sequence may occur in `.fasta` file several times, in that case, occurrences are counted and the count saved.

Header lines may contain more information - sequence count and labels. In that case, header lines will contain fields separated by `|` sign. The second field is considered as sequence count and the third field as label, if present. Forth and any following fields are ignored. One fasta header may only contain one label. If more labels for the same sequence are needed, sequence must be repeated several times - once for each label.

Example of valid .fasta file:

```
>
WVTAPRSLPVLP
>4863
GSWVVDISNVED
>4628|8
NYSGNRPLPGIW
>6642|4|label1
RSPIVRQLPSLP
>6643|3|label2|something
RSPIVRQLPSLP
>664
RSPIVRQLPSLP
>4893|1|label2
AKSRPLPMVGLV
```

Table 5.1.1 shows sequences and their counts in particular labeled groups resulting from such .fasta file:

sequence	label1	label2	no_label
WVTAPRSLPVLP	0	0	1
GSWVVDISNVED	0	0	1
NYSGNRPLPGIW	0	0	8
RSPIVRQLPSLP	4	3	1
AKSRPLPMVGLV	0	1	0

Table 1: Sequence and label counts resulting from example file parsing

.tsv (.csv) format The second option of sequence input is a .tsv table with exactly the same structure as table 5.1.1.

All fields must be separated by tab characters, the header line must contain all the labels. Sequences must be in the first column, the name of the first column may be arbitrary. Each unique sequence must be present on exactly one line, if a sequence is present in 0 copies in some label group, a cell containing 0 must be present - i.e. all lines must have the same number of tab-separated fields.

Example

```
sequence\tlabel1\tlabel2\tno_label
WVTAPRSLPVLP\t0\t0\t1
GSWVVDISNVED\t0\t0\t1
NYSGNRPLPGIW\t0\t0\t8
RSPIVRQLPSLP\t4\t3\t1
AKSRPLPMVGLV\t0\t1\t0
```

\t stays for tab character. This file describes exactly the same dataset as previous table 5.1.1.

5.1.2 Cluster input

In `cluster` mode, Hammock accepts an input file containing clusters previously generated by a Hammock run in `greedy`, `clinkage`, `full` or `cluster` mode. Such file must be in the .tsv table format identical to output `_sequences.tsv` files. See 5.2.1 for details.

5.2 Output files

Hammock outputs several files into a directory controlled by `-d, --outputDirectory` parameter (see 8.3).

5.2.1 Types of files saved

Hammock represents sets of resulting clusters in three differnet ways: by `_sequences.tsv` files, `_sequences_original_order.tsv` and `_clusters.tsv` files. In addition, multiple sequence alignments, remaining sequences from the sequence pool, input statistics and run logs are saved.

files with _sequences.tsv extension tsv (tab separated) table files. Such file contains full list of sequences, one line per sequence. For each sequence, cluster membership, alignment (if available) and label counts are listed. Clusters are sorted by size starting from the largest, in every cluster, sequences are sorted by sum of occurrences with all labels, starting from the most frequent. File contains header line.

Example:

```
cluster_id\tsequence\talignment\tsum\tround1\tround2
2\tEVMSTSDLHRLS\t--EVMSTSDLHRLS-\t51\t11\t40
2\tETDAYTDLHRLA\t---ETDAYTDLHRLA\t28\t9\t19
2\tIGSQSDLHKLTI\t---IGSQSDLHKLTI\t5\t4\t1
2\tEHDMTGYSDLWR\tEHDMTGYSDLWR---\t2\t0\t2
1\tTTMWPPSLNLPL\t-TTMWPPSLNLPL--\t10\t4\t6
1\tTTPPGVHSLAPV\t---TTPPGVHSLAPV\t8\t2\t6
1\tTTSYWWPQNHMD\tTTSYWWPQNHMD---\t3\t3\t0
1\tTTYTPTLHGIMP\t--TTYTPTLHGIMP-\t3\t1\t2
1\tTVRPPLMSAWLV\t--TVRPPLMSAWLV-\t1\t1\t0
```

\t stays for tab character. File contains two clusters, one of size 86 containing 4 unique sequences, the other of size 25, containing 5 unique sequences.

files with _sequences_original_order.tsv extension tsv (tab separated) table files. Such file contains full list of sequences, one line per sequence. It is very similar to files with _sequences.tsv, with two differences. First, sequences are not ordered according to resulting clusters, but according to the original order in the input files used. If .fasta file was used with more than one occurrence of a sequence, the first occurrence defines the order. Second, this file also contains sequences not belonging to any cluster (those left in the sequence pool after the clustering process). Such sequences have NA in their corresponding cluster_id and alignment fields. These files are only saved in greedy, clinkage and full modes (i.e. in modes where single sequences, not clusters were used as input).

Example:

```
cluster_id\tsequence\talignment\tsum\tround1\tround2
2\tEVMSTSDLHRLS\t--EVMSTSDLHRLS-\t51\t11\t40
1\tTVRPPLMSAWLV\t--TVRPPLMSAWLV-\t1\t1\t0
2\tETDAYTDLHRLA\t---ETDAYTDLHRLA\t28\t9\t19
NA\tETDARRDLHRLA\tNA\t5\t6\t2
2\tEHDMTGYSDLWR\tEHDMTGYSDLWR---\t2\t0\t2
1\tTTMWPPSLNLPL\t-TTMWPPSLNLPL--\t10\t4\t6
1\tTTSYWWPQNHMD\tTTSYWWPQNHMD---\t3\t3\t0
NA\tRTSYAAPQNAMD\tNA\t9\t0\t11
2\tIGSQSDLHKLTI\t---IGSQSDLHKLTI\t5\t4\t1
1\tTTYTPTLHGIMP\t--TTYTPTLHGIMP-\t3\t1\t2
1\tTTPPGVHSLAPV\t---TTPPGVHSLAPV\t8\t2\t6
```

\t stays for tab character. File contains two clusters, one of size 86 containing 4 unique sequences, the other of size 25, containing 5 unique sequences and two sequences not belonging to any cluster.

files with _clusters.tsv extension tsv (tab separated) table files. Such file contains full list of clusters, one line per cluster. For each cluster, id, the most frequent sequence and sums of sequence occurrences for every label are listed. Lines are sorted by cluster size, starting from the largest cluster. File contains header line. This file may be used in downstream analysis e.g. to generate a heatmap.

Example:

```
cluster_id\tmain_sequence\tsum\tround1\tround2
2\tEVMSTSDLHRLS\t86\t24\t62
1\tTTMWPPSLNLPL\t25\t11\t6\t14
```

\t stays for tab character. File contains information on the same two clusters as in previous example.

File final_remaining_sequences.fa This file contains all the sequences not belonging to any cluster (those left in the sequence pool after the clustering process). It is in the fasta format described in section 5.1.1. This file is only saved in `cluster` and `full` modes.

Files with .aln extension These files contain multiple sequence alignments in aligned fasta format, one file per cluster.

input_statistics.tsv This file contains information on input dataset label counts. Saved in every run.

run.log Contains copy of what's outputted to stderr. Saved in every run.

5.2.2 Files saved for runs in particular modes

results saved in `greedy` and `clinkage` modes In initial clustering modes, resulting clusters are aligned according to greedy clustering or Clustal for `clinkage`. Three files are saved, all representing the same result in different ways: `initial_clusters_sequences.tsv`, `initial_clusters_sequences_original_order.tsv`, `initial_clusters_sequences.csv` `initial_clusters.tsv`. See 5.1 for details on file formats). File `initial_clusters_sequences.tsv` may be directly used as input for a run in `cluster` mode.

results saved in `cluster` mode `cluster` mode gets aligned clusters on input. Resulting final clusters are saved in files `final_clusters.tsv`, `final_clusters_sequences.tsv` and `final_remaining_sequences.fa`.

MSAs for initial and final clusters are saved in aligned fasta format in folders `alignments_intial` and `alignments_final` folders. MSAs generated throughout the process of extending and merging clusters in several rounds (see 4 for details) are saved in `alignments_other` folder.

results saved in `full` mode In `full` mode, all files saved in both `greedy` or `clinkage` and `cluster` modes are saved. In addition, file `final_clusters_sequences_original_order.tsv` is saved.

6 Examples

In this section, we will demonstrate the usage of Hammock on sample dataset. Example data and expected results are stored in examples folder.

6.1 Example 1: MUSI

This dataset was originally published with the MUSI tool [4]. It contains sequences describing transcription factor binding sites. This dataset is suitable for example, because it is very small, consists of short sequences (obtained by Phage display) and contains very clear motifs.

6.1.1 Start

First, we move to appropriate folder (Suppose `Hammock` is placed in home folder).

```
$ cd ~/Hammock/examples/MUSI/
```

We start by typing in the command from quick start 8.1.

```
$ java -jar ../../dist/Hammock.jar full -i musi.fa
```

It runs Hammock in `full` mode, leaving all parameters set to default values.

When the execution finishes, a folder called `Hammock_result_1` appers in `dist` folder. Folder contains several files and subfolders. To see brief overview of the results, we can inspect file `final_clusters.csv`.

```
$ cd ../../dist/Hammock_result_1
$ column -s "\t" -t final_clusters.tsv
cluster_id  main_sequence  sum   no_label
4330        AAMFLRPLPAVQ   1670  1670
4334        AALPKLPFRNMT   311   311
4407        GSWAVDISNVED   12    12
```

We can see that Hammock identified 3 clusters. They are too big for manual inspection of their MSAs, but we can still have a look

```
$ column -s "\t" -t final_clusters_sequences.tsv | less
```

In the first cluster, central "LP" motif is clearly visible, but we would still like to see some better visualization. In `alignments_final` folder, there are cluster MSA's, one file per cluster. We can use them e.g. to generate sequence logos. Suppose we have installed weblogo 3.4 [1] before.

```
$ mkdir logos_final
$ for file in alignments_final/*.aln
> do
> echo $file
> weblogo --format PNG --sequence-type protein --size large --errorbars NO \
> --resolution 299 --composition equiprobable --color-scheme chemistry < $file > \
```

```
>  "logos_final/$(basename "$file" .aln)".png
> done
```

The three .png files in `logos_final` folder show strong motifs conserved in our clusters. The smallest cluster seems like a it gathers just a few versions of the same sequence. We believe these sequences are "parasites", i.e. noise present due to the phenomenon of unbalanced phage amplification.

In `alignments_other` folder, there are alignment files from the entire iterative cluster extension and merging process.

There are 464 sequences in `final_remaining_sequences.fa` that were not assigned to any cluster (The information on this count is also stored in `run.log`). We'll try to change parameters, so that more sequences are assigned to clusters.

In `run.log`, we can see that 3 rounds of cluster extension and merging were performed and that assign threshold values were absolute and set to: `11.4,9.0,6.6`. We'll run Hammock again with lower threshold values of `11.4, 8.5, 6.0`. As greedy clustering results are ok, we'll re-use them and run Hammock in `cluster` mode[1]. This time, we'll also use 8 computational threads. The resulting command look like this:

```
cd ~/Hammock/
java -jar dist/Hammock.jar cluster \
-i dist/Hammock_result_1/initial_clusters_sequences.tsv \
--assign_thresholds 11.4,8.5,6.0 \
-t 8
```

We can see that with these parameters, there are only 378 sequences remaining.

7 Galaxy implementation

Note: The Galaxy implementation takes a little longer to update, so when a new version of Hammock is published, the Galaxy implementation might be still unsing an older version for a while. The version in use is stated in the Galaxy repository.

Hammock is also available as a tool for the Galaxy toolbox [3]. The Galaxy version contains slightly less functionality than full standalone version. This section states the differences between these two versions and provides details on the Galaxy version.

[1] In this case, greedy clustering takes so short that it is not actually necessary, but for large datasets and intensive parameter testing, this can save significant amount of time.

7.0.1 Download

The Galaxy version of Hammock is available for download and installation from the main tool shed at `https://toolshed.g2.bx.psu.edu/` in repository called *hammock* in category *Sequence Analysis*.

Direct link: `https://toolshed.g2.bx.psu.edu/view/hammock/hammock/d90f4809ccc6`

7.0.2 Galaxy version input

The Galaxy version only supports fasta input format, see 5.1.1 for details.

7.0.3 Galaxy version outputs

The Galaxy version only outputs two files, both of them are .tsv tables. The `sequences.tsv` file contains detailed information on all sequences contained in resulting clusters (see 5.2.1 for details). The `clusters.tsv` file contains a less detailed summary on resulting clusters (see 5.2.1 for details).

7.0.4 Galaxy run parameters

The Galaxy version only supports runs in `full` mode. See 8.3 and 8.4 for detailed description of parameters. The Galaxy version supports all of the parameters mentioned in these sections, except `-d, --outputDirectory` (output is handled by Galaxy), `-f, --file format` (only .fasta format is supported) and `-t, --threads` (threads are handled by Galaxy).

8 Manual pages

8.1 Quick start

```
java -jar ~/Hammock/dist/Hammock.jar full -i ~/Hammock/examples/musi.fa
```

8.2 Synopsis

Hammock is packed as an executable java .jar archive. It can be invoked as follows:

`java -jar` *jvm_args* `Hammock.jar` *mode param1 param2 param3...*

where *mode* is one of following:

```
full
greedy
clinkage
cluster
```

8.3 Parameters common for all modes

These parameters may be used when any of 4 possible modes is invoked.

`-d, --outputDirectory` ⟨*directory*⟩ All output files will be saved into this directory. If the directory does not exist, it will be created. If it does exist, Hammock will stop to prevent overwriting.

Default: directory `Hammock_result_n` in `dist` folder will be created. By default, `n` is 1, if `Hammock_result_1` exists, then `n` is 2 etc.

`-t, --threads` ⟨*int*⟩ Number of threads to be used for computation.

Default: 4.

`-l, --labels` ⟨*str,str,str...*⟩ Only sequences carrying these labels will be considered. This parameter also defines the order of labels in all output files. See 5.1.1 for details about labels.

Default: All labels are used. The order in output files is from the most abundant label to the least abundant.

`--temp` ⟨*directory*⟩ A directory to store the temporal files in. Note that this option is overridden by the settings.prop file, if corresponding lines are present. The option is useful especially in supercomputing environments when appropriate temporal files storage paths are allocated dynamically. The directory has to be writable and faster file systems are preferred, as Hammock might generate many small temporal files.

Default: /tmp

8.4 Parameters specific for `full` mode

In this mode, Hammock `full` first invokes *initial clustering* procedure, than uses its results as input for *hmm clustering* procedure. See section 4 for more details.

All the parameters specific for both `greedy/clinkage` and `cluster` modes are allowed with the `full` mode. Parameter `-i, --input` behaves the same way as when used with `greedy/clinkage` mode, i.e. expects a sequence file the type of which can be specified by `-f, --file_format` parameter. See 8.5 for details.

`--use_clinkage` If this switch is present, clinkage clustering will be used as the initial step.

Default: By default, clinkage clustering is used when there are up to 10 000 unique input sequences.

`--use_greedy` If this switch is present, greedy clustering will be used as the initial step.

Default: By default, greedy clustering is used when there are more than 10 000 unique input sequences.

8.5 Parameters specific for initial clustering modes (`greedy` and `clinkage`)

`-i, --input` ⟨*file*⟩ **Required.** A path to input file containing input sequences in one of supported formats. Expected file format is controlled by `-f, --file_format` parameter.

Default: No default. This parameter is required.

`-f, --file_format` ⟨*[fasta, tab]*⟩ Expected format of file specified by `-i, --input` parameter. One of two values is expected:

`fasta` Sequences in fasta format

`tab` Sequences in .csv table format

See section 5 for information on file formats.

Default: `fasta`

`-m, --matrix` ⟨*file*⟩ A path to a file containing amino-acid substitution matrix. A decent selection of matrices is provided in `Hammock/matrices/` folder. Any other matrix respecting the same file format may be provided

Default: `Hammock/matrices/blosum62.txt`

`-g, --alignment_threshold` ⟨*int*⟩ Minimal score needed for a sequence to join a cluster during initial clustering. For both the clustering routines, each sequence

in a cluster must reach at least this score to all the other sequences in the cluster. Only positive integer values are allowed. For compatibility reasons, alias `--greedy_threshold` can also be used for this parameter.

Default: If not set, threshold value is determined based on average sequence length. The value is $1.7 \times average\_seq\_legth$ rounded to integer.

`-x, --max_shift` ⟨*int*⟩ The size of maximal sequence-sequence shift. During initial clustering, Hammock searches for best sequence-sequence alignment without inner gaps. This parameter specifies the maximal number of positions by which sequences are allowed to be shifted during alignment. Higher values may mean a bit more sensitivity for the price of slower computation. Non-negative integer values only. The maximal value allowed is the length of the shorted sequence -1. If a higher value is set, Hammock will adjust it to the maximal.

Default: If not set, the value is determined on the basis of average sequence length. The value is $0.25 \times average\_seq\_legth$ rounded to integer.

`-p, --gap_penalty` ⟨*-int*⟩ During greedy clustering, Hammock searches for best sequence-sequence alignment without inner gaps. This parameter specifies the negative penalty given to score of an alignment for each amino acid aligned towards a (trailing) gap. Non-positive integer values only.

Default: 0

8.6 Parameters specific for `greedy` mode

`-R, --order` ⟨*[size, alphabetic, random, input]*⟩ This parameter specifies the order of sequences during greedy clustering. One of three values is expected:

`size` Sequences are sorted from the one with the most copies to the one with the least. Sequences with equal numbers of copies are sorted alphabetically.

`alphabetic` Sequences are sorted alphabetically.

`random` Random order of sequences is used. In order to achieve determinism and reproducibility, the random order depends on the `-S, --seed` parameter

`input` Order from the input file is preserved.

Default: `size`

`-S, --seed` ⟨*int*⟩ The seed to be used as the base for pseudorandom sequence order. Only applicable when `-R random` is in use. For a defined seed, the order (and so the result of greedy clustering) is deterministic.

Default: 42

--initial_clusters_limit ⟨*int*⟩ Simplifying a bit, we can say that the greedy clustering algorithm takes sequences one-by-one, starting from the beginning of the list (defined by `-R, --order` parameter) and builds a cluster around each of them. This number specifies how many sequences will get the chance to have a cluster built around. It is the maximal number of clusters of size more than 1 that will result from greedy clustering.

Default: 2.5 % of the number of unique sequences in the dataset.

8.7 Parameters specific for `clinkage` mode

-L, --cache_size_limit ⟨*int*⟩ Size threshold for a cluster have the scores to other clusters cached. Normally, there should be no need to change this parameter. In case `clinkage` mode is used with large datasets, Hammock may require unacceptable amounts of memory. Increasing this parameter lowers the demand for memory, but also increases the execution time. Reasonable values should be approximately from 1 to 10, the use of higher values practically turns off any caching.

Default: 1

8.8 Parameters specific for `cluster` mode

-i, --input ⟨*file*⟩ **Required.** A path to an input file containing clusters in .tsv table format (if produced by Hammock, the files with _sequences.tsv extension).

Default: No default. This parameter is required.

-as, --additional_sequences ⟨*file*⟩ A path to an input file containing sequences in fasta format. These sequences will be added to the sequence pool. Especially useful when additional extension (and merging) rounds are needed. If this is the case, use this parameter to include the `final_remaining_sequences.fa` file from a finished run to continue exactly as if the run was specified to perform more extension/merging rounds.

Default: No default.

-U, --unique If this switch is present, initial clusters will be ordered by their unique size, i.e. the number of unique sequences they contain, rather than by their total size, i.e. the sum of all sequence counts across all labels. The order defines which clusters will be selected as cluster cores for HMM-based clustering (also influenced by `-c, --count`.

Default: By default, this switch is OFF.

-c, --count_threshold ⟨*int*⟩ This many largest initial clusters will be used as cluster cores for HMM-based clustering. Clusters are ordered by total size or unique

size, depending on `-U, --unique` parameter. The result will never contain more than this number of clusters.

Default: 2.5 % of the total number of initial clusters, with the maximum of 250 and minimum of 25 cores (manual setting overrides maximum and minimum limits).

`-E, --initial_extension_threshold` ⟨*float*⟩ Minimal cluster-cluster similarity needed for a small cluster to be merged with a cluster core during the initial extension. Negative values will cause Hammock to skip the initial cluster extension step.

Default: The first value of `-r, --merge_thresholds` ×1.1.

`-n, --assign_thresholds` ⟨*float,float,float...*⟩ Length of this sequence of thresholds defines the number of clustering rounds. Each threshold value is used as the minimal score (inclusive) returned by `hmmsearch` needed for a sequence to be added into a cluster in the respective extension step. If a negative value is found, the corresponding extension step is skipped (and the following merging step is run immediately).

Default: By default, 3 rounds of clustering will be performed. Threshold values are determined based on average input peptide sequence length. If scores are absolute (switch `-b, --absolute_thresholds` is ON (default)) thresholds are definded as: $(0.95, 0.75, 0.55) \times average\_seq\_legth$. If scores are relative, (switch `-e, --relative_thresholds` is ON), thresholds are defined as: $(0.13, 0.113, 0.108) \times average\_seq\_legth$.

`-v, --overlap_thresholds` ⟨*float,float,float...*⟩ For each round of clustering, specifies minimal score reported by `hmmsearch` needed for a sequence to be assigned into overlapping sets of sequences. Clusters having overlapping sets of found sequences will be compared using hmm-hmm alignment and may possibly be merged. Setting lower values may result in increase in sensitivity (more cluster merging) for the price of greater resource consumption. On the other hand, large values will prevent clusters from being merged. When the threshold is 0, all clusters will be compared and possibly merged if similar enough.

The length of this threshold sequence must be the same as the length of sequence specified by `-n, --assign_thresholds` parameter.

Default: By default, 3 rounds of clustering will be performed. Threshold values are determined based on average input peptide sequence length. If scores are absolute (switch `-b, --absolute_thresholds` is ON (default)) thresholds are definded as: $(0.7, 0.4, 0.0) \times average\_seq\_legth$. If scores are relative, (switch `-e, --relative_thresholds` is ON), thresholds are defined as: $(0.09, 0.075, 0.0) \times average\_seq\_legth$. If `-n, --assign_thresholds` parameter is set, so that the number of clustering rounds is not equal 3, whole sequence of

thresholds is determined based on `-n, --assign_thresholds` parameter as: $(assign\_thresholds) \times 0.75$ with the last score set to 0.

It is generally recommended to set at least the last score to 0 and at least the first score to some reasonable positive value.

`-r, --merge_thresholds` $\langle float,float,float... \rangle$ For each cluster merging step, specifies the minimal score reported by `hhalign` needed for two clusters to be merged. Only clusters having overlapping sets of found sequences are tested and possibly merged. The length of this score sequence must be the same as the length of sequence specified by `-n, --assign_thresholds` parameter. If a negative value is found, the corresponding cluster merging step is skipped and the following extension step is run immediately.

Default: By default, 3 rounds of clustering will be performed. Threshold values are determined based on average input peptide sequence length. If scores are absolute (switch `-b, --absolute_thresholds` is ON (default)) thresholds are definded as: $(0.95, 0.75, 0.55) \times average\_seq\_legth$. If scores are relative, (switch `-e, --relative_thresholds` is ON), thresholds are defined as: $(0.125, 0.115, 0.110) \times average\_seq\_legth$. If `-n, --assign_thresholds` parameter is set, so that the number of clustering rounds is not equal 3, whole sequence of thresholds is determined based on `-n, --assign_thresholds` parameter. Thresholds are defined as: $(assign\_thresholds) \times 1.0$.

`-e, --relative_thresholds` If this switch is present, all scores defined by `-n, --assign_thresholds`, `-v, --overlap_thresholds` and `-r, --merge_thresholds` are interpreted as per hmm match state scores. It means that absolute threshold value is computed as ($value \times length$ where *value* is the value set by parameters and *length* is the number of match states of shorter hmm (when hmm vs. hmm are compared) or the length of sequence (when hmm vs. sequence are compared and sequence is shorter than the number of hmm match states). This is the opposite from `-b, --absolute_thresholds` switch.

Default: By default, this switch is OFF.

`-b, --absolute_thresholds` If this switch is present, all scores defined by `-n, --assign_thresholds`, `-v, --overlap_thresholds` and `-r, --merge_thresholds` are interpreted as absolute threshold values. It means that these numbers are directly used in computation and no length correction is made. This is the opposite from `-e, --relative_thresholds` switch.

Default: By default, this switch is ON.

`-h, --min_conserved_positions` $\langle int \rangle$ At any stage of the algorithm, clusters are not allowed to have less than this number of conserved positions (i.e. conserved

multiple alignment columns without too many gaps). If this option is set to 0, no conserved positions control is performed.

Default: If not set, the value is determined on the basis of average sequence length. The value is $0.33 \times average\_seq\_legth$ rounded to integer.

-y, --max_gap_proportion ⟨*float*⟩ Defines maximal proportion of gaps in conserved positions. Any multiple sequence alignment column containing more gaps will not be considered a conserved position.

Default: 0.2

-k, --min_ic ⟨*float*⟩ Defines minimal information content a conserved position (in terms of the Shannon entropy). Note that maximal information content (meaning mulitple alignment column containing the same amino acid in all rows) is 4.3219.

Default: 1.2

-j, --max_aln_length ⟨*int*⟩ Defines maximal alignment length. If a sequence insertion or cluster merging event would result in a cluster having multiple sequence alignment with more positions than this value, the insertion/merging is not performed.

Default: If not set, threshold value is determined based on average sequence length. The value is $2.0 \times average\_seq\_legth$ rounded to integer.

-u, --max_inner_gaps ⟨*int*⟩ Defines maximal number of inner (non-trailing) gaps for each line of each multiple sequence alignment. If this parameter is more than 0, it might cause a non-match state to exist between 2 match states, representing insertion/deletion in a part of the cluster sequences.

Default: 0

-q, --extension_increase_length If this switch is present, sequence insertion routine is allowed to insert sequences in a cluster, even if such insertion increases the number of positions in cluster's multiple sequence alignment.

Default: By default, this switch is OFF.

-C, --min_correlation ⟨*float[-1.0, 1.0]*⟩ This parameter only makes sense when multiple sequence labels are in use. Defines the minimal value of Pearson correlation coefficient between the vectors of sequence-label abundances of two clusters to be merged or a sequence and a cluster when the sequence is about to join the cluster. If the threshold is not reached, the merging/joining is not performed. The value of -1.0 implies no such constraint is applied.

Default: -1.0

9 Other settings

9.1 The `settings.prop` file

This file contains options less often needed to be changed. It defines commands for calling external tools, paths to folders containing temporal files and parameters for external tools.

9.1.1 File format

Lines starting with hash (#) character ale ignored.
Each option is defined as a line containing pair *key=value* where *key* is option name.

9.1.2 Commands for external tools

By default, commands (paths to) external tools need not to be specified. If a command is not specified, Hammock expects to find appropriate file in appropriate folder according to original folder structure Hammock comes with. Paths are internally defined relative to `Hammock.jar` file in `dist` folder. Namely:

Clustal Omega file as: `Hammock/clustal-omega-1.2.0/clustalO-64bit`
Hmmer's hmmbuild file as: `Hammock/hmmer-3.1b1/src/hmmbuild`
Hmmer's hmmsearch file as: `Hammock/hmmer-3.1b1/src/hmmsearch`
HHSuite's hhmake file as: `Hammock/hhsuite-2.0.16/bin/hhmake`
HHSuite's hhsearch file as: `Hammock/hhsuite-2.0.16/bin/hhsearch`

9.1.3 Paths to temporal files

During each run, Hammock generates and deletes several types of temporal files. By default, all temporal files are placed in subfolders of `/tmp/Hammock_temp_[time]` folder. The location of the master temporal files directory can be changed using the `--temp` switch. Also, in `settings.prop` file, it is possible to change the master temporal directory, as well as to specify locations of particular temporal subdirectories separately (e.g. the directory where MSAs are stored). Any settings present (uncommented) in the settings.prop file overrides the `--temp` switch. Note that if temporal subdirectories already exist at the beginning of a Hammock run, their contents are deleted. Be sure to set the temp directory path so that the directory is writable.

Hammock may generate up to thousands of (generally very small) temporal files in each run, depending on input size and complexity.

9.1.4 Parameters for external tools

It is generally not recommended to change anything in this section. Some changes may result in Hammock not working at all, others may have strong impact on results. See manuals of external tools for definitions of parameters used.

9.2 jvm parameters

For big datasets, it may be necessary to allow jvm to allocate more memory. This can be done using `-Xmx` parameter, e.g. `-Xmx4g` to allocate up to 4 gigabytes.

References

[1] G. E. Crooks. WebLogo: A sequence logo generator. *Genome Research*, 14(6):1188–1190, may 2004. doi: 10.1101/gr.849004. URL https://doi.org/10.1101/gr.849004.

[2] R. D. Finn, J. Clements, and S. R. Eddy. HMMER web server: interactive sequence similarity searching. *Nucleic Acids Research*, 39(suppl):W29–W37, may 2011. doi: 10.1093/nar/gkr367. URL http://dx.doi.org/10.1093/nar/gkr367.

[3] B. Giardine. Galaxy: A platform for interactive large-scale genome analysis. *Genome Research*, 15(10):1451–1455, sep 2005. doi: 10.1101/gr.4086505. URL https://doi.org/10.1101/gr.4086505.

[4] T. Kim, M. S. Tyndel, H. Huang, S. S. Sidhu, G. D. Bader, D. Gfeller, and P. M. Kim. MUSI: an integrated system for identifying multiple specificity from very large peptide or nucleic acid data sets. *Nucleic Acids Research*, 40(6):e47–e47, dec 2011. doi: 10.1093/nar/gkr1294. URL https://doi.org/10.1093/nar/gkr1294.

[5] Adam Krejci, Ted R. Hupp, Matej Lexa, Borivoj Vojtesek, and Petr Muller. Hammock: a hidden markov model-based peptide clustering algorithm to identify protein-interaction consensus motifs in large datasets. *Bioinformatics*, page btv522, sep 2015. doi: 10.1093/bioinformatics/btv522. URL https://doi.org/10.1093/bioinformatics/btv522.

[6] F. Sievers, A. Wilm, D. Dineen, T. J. Gibson, K. Karplus, W. Li, R. Lopez, H. McWilliam, M. Remmert, J. Soding, J. D. Thompson, and D. G. Higgins. Fast, scalable generation of high-quality protein multiple sequence alignments using clustal omega. *Molecular Systems Biology*, 7(1):539–539, jan 2011. doi: 10.1038/msb.2011.75. URL http://dx.doi.org/10.1038/msb.2011.75.

[7] J. Soding. Protein homology detection by HMM-HMM comparison. *Bioinformatics*, 21(7):951–960, nov 2004. doi: 10.1093/bioinformatics/bti125. URL https://doi.org/10.1093/bioinformatics/bti125.