USER GUIDE

ChIPMunk

MOTIF DISCOVERY IN CHIP-SEQ DATA

http://autosome.ru/ChIPMunk/

CONTENTS

Summary

ChIPMunk is the motif discovery tool for huge sequence datasets. ChIPMunk is based on a greedy approach accompanied by subsampling. Originally, ChIPMunk was designed to produce positional weight matrices (position weight matrices, PWMs) for transcription factor binding sites by integrating large sequence datasets from different sources including ChIP-chip data [1]. Then it was adapted [2] to properly handle ChIP-Seq data and account for motif positional preferences reflected by ChIP-Seq base coverage profiles (ChIP-Seq peak shapes). ChIPMunk was integrated in various software tools, see e.g. http://BioUML.org, http://MotifLab.org and http://Nebula.curie.fr. We used ChIPMunk to construct HOCOMOCO (Homo Sapiens Comprehensive MOtif COllection, see http://autosome.ru/HOCOMOCO/); see [3] for discussion of other ChIPMunk applications.

ChIPMunk uses Kullback Discrete Information Content, KDIC, as the measure for motif quality. The Discrete Information Content concept was firstly introduced with BigFoot, tool for gapless multiple local alignment of DNAse footprints [4]. In this updated version (v6, 2014) ChIPMunk software includes its dinucleotide progeny, diChIPMunk [5], which uses the same approach to construct dinucleotide positional weight matrices (diPWMs) accounting for dinucleotide composition, significantly improving recognition of binding sites [6]. Kullback Dinucleotide Discrete Information Content, KDIDIC, is used in diChIPMunk as KDIC in ChIPMunk.

ChIPMunk is implemented in Java (1.6) and efficiently handles sequence sets of hundreds and thousands of sequences on a modern desktop or laptop.

[1] Kulakovskiy and Makeev, *Biophysics*, 2009, http://www.ncbi.nlm.nih.gov/pubmed/20067172

[2] Kulakovskiy et al., *Bioinformatics*, 2010, http://www.ncbi.nlm.nih.gov/pubmed/21852305

[3] Kulakovskiy and Makeev, *APCSB*, 2013, http://www.ncbi.nlm.nih.gov/pubmed/23790213

[4] Kulakovskiy et al., *Bioinformatics*, 2009, http://www.ncbi.nlm.nih.gov/pubmed/19605419

[5] Kulakovskiy et al., *JBCB*, 2013, http://www.ncbi.nlm.nih.gov/pubmed/23427986

[6] Levitsky et al., *BMC Genomics*, 2014, http://www.ncbi.nlm.nih.gov/pubmed/24472686

Citing ChIPMunk

If you want to cite ChIPMunk in your research please refer to [2] for the basic mononucleotide version and to [5] for the dinucleotide version. The full references are:

Deep and wide digging for binding motifs in ChIP-Seq data. Kulakovskiy IV, Boeva VA, Favorov AV, Makeev VJ. *Bioinformatics*. 2010 Oct 15;26(20):2622-3. doi: 10.1093/bioinformatics/btq488. Epub 2010 Aug 24.

From binding motifs in ChIP-Seq data to improved models of transcription factor binding sites. Kulakovskiy I, Levitsky V, Oshchepkov D, Bryzgalov L, Vorontsov I, Makeev V. *J Bioinform Comput Biol*. 2013 Feb;11(1):1340004. doi: 10.1142/S0219720013400040. Epub 2013 Jan 16.

Asking questions

If you have any questions regarding ChIPMunk please don't hesitate to contact Ivan Kulakovskiy:

```
ivan-dot-kulakovskiy-at-gmail-dot-com
```

A note on terminology

Here we use the term "motif" to denote the set of similar words, i.e. the pattern. The "motif occurrences" or "motif hits" denote particular substrings matching the detected motif in the input data.

Quick-start guide

Prerequisites

ChIPMunk is implemented in Java (1.6) and requires the Java Runtime Environment (1.6 or more recent) to be installed. You can check whether you already have proper JRE installed by running

```
java -version
```

from the command line. Something like

```
java version "1.7.0_21"
```

in response would perfectly fit your needs.

On Linux: you may already have JRE installed, or can install it from the respective package repository. Some time ago ChIPMunk was faster when running on Sun/Oracle JDK versus OpenJDK, but there should be no significant differences today.

On Windows: visit http://java.com and follow the instructions.

Running ChIPMunk

Suppose you have the sequences with putative "motif" (a set of overrepresented words similar to some yet unknown pattern) at hand in a basic multi fasta file named **`sequences.mfa`**.

Suppose you unpacked ChIPMunk into your working directory and you see **`chipmunk.jar`** file there. Then you can simply call ChIPMunk:

```
java -jar chipmunk.jar s:sequences.mfa
```

and get the primary major motif from the 7 to 22 nucleotides motif lengths range.

This would be OK for strong motifs and small datasets; this is also OK to test whether ChIPMunk is working properly. However, you will need deeper understanding of input data, parameters and results to discover motifs in complex datasets, so do not hesitate to look for more details below.

The same result should be achieved by running

```
java -cp chipmunk.jar ru.autosome.ChIPMunk s:sequences.mfa
```

i.e. explicitly specifying the main Java class (ru.autosome.ChIPMunk) and using the jar-file in the Java class path (-cp).

This will allow running diChIPMunk as well:

```
java -cp chipmunk.jar ru.autosome.di.ChIPMunk s:sequences.mfa
```

If executed without proper arguments (e.g. not specifying the set of sequences) ChIPMunk should print something like:

```
ru.autosome.ChIPMunk usage: <start_length> <stop_length>
<verbose=(n)o|(y)es> <mode=(o)ops|zoops_factor=1.0|0.0>
<ru.autosome.ChIPAct parameters>

Exception in thread "main" java.lang.RuntimeException: not enough
parameters

        at ru.autosome.ChIPMunk.<init>(ChIPMunk.java:113)

        at ru.autosome.ChIPMunk.main(ChIPMunk.java:33)
```

ChIPMunk prints all the results into the standard output stream (STDOUT); all the messages are written to the standard error stream (STDERR). Thus typically you'd like to store the results in a file and see the messages on-screen:

```
java -cp chipmunk.jar ru.autosome.ChIPMunk s:sequences.mfa
>results.txt
```

Another way is to save both the log and the results:

```
java -cp chipmunk.jar ru.autosome.ChIPMunk s:sequences.mfa
1>results.txt 2>log.txt
```

Memory requirements

To analyze huge sequence ChIPMunk may require more memory. In practice it's enough to allocate a java heap from 512Mb to 1Gb to analyze most of the real-life datasets. For the Oracle Java VM it can be done using the following switches:

```
java -Xms512M -Xmx1G
```

The command-line will look like:

```
java -Xms512M -Xmx1G -cp chipmunk.jar ru.autosome.ChIPMunk
```

for ChIPMunk and

```
java -Xms512M -Xmx1G -cp chipmunk.jar ru.autosome.di.ChIPMunk
```

for diChIPMunk.

Putting everything together

Running ChIPMunk in default mode using simple multifasta file and from 512Mb to 1 Gb of memory:

```
java -Xms512M -Xmx1G -cp chipmunk.jar ru.autosome.ChIPMunk
s:sequences.mfa 1>results.txt 2>log.txt
```

The same for diChIPMunk:

```
java -Xms512M -Xmx1G -cp chipmunk.jar ru.autosome.di.ChIPMunk
s:sequences.mfa 1>results.txt 2>log.txt
```

A note on ChIPMunk versions

During several years of development ChIPMunk received many changes, including bugfixes and tweaks affecting internal parameter settings or results representation. This guide covers the most recent version of ChIPMunk. The most significant change is the KDIC/KDIDIC approximate scaling to [0..1] range introduces in this version. This scaling does not affect the results but makes it easier for the user to compare the overall quality of motifs/alignments/etc. The original scale was not normalized.

General workflow overview

ChIPMunk iteratively looks for the most significant gapless multiple local alignment (GMLA) using the Discrete Information Content (with the Kullback term) as a measure for the alignment quality. The best GMLA has the highest KDIC (or K-dinucleotide-DIC in the case of the dinucleotide version).

The optimal GMLA is constructed using an iterative optimization of starting positional weight matrices (PWM), seeds, which are either randomly generated (default) or derived from a given set of seeding sequences. At each iteration step, ChIPmunk searches for the best PWM hits in all sequences and re-evaluates PWM from those best hits. To avoid being stuck in a local optimum ChIPMunk uses optimization on random subsets. To speed-up the convergence to a proper solution ChIPMunk uses a global storage for a current optimal solution which is accessible for all computational threads and can be used as a starting point in the case a solution from a seeding PWM was too bad. Please see the 'formal math' section for a flowchart and a pseudocode representation.

If zero-or-one-occurrence-per-sequence, ZOOPS, mode was used, then ChIPMunk segregates the best alignment into "background" and "motif" components, selecting only the good-scoring words into the motif.

ChIPMunk iteratively checks a given motif lengths range looking for the longest motif passing the "strongness" criteria, i.e. carrying the flanking alignment columns with the information content passing a minimal threshold (please see the "math" section for details below).

ChIPHorde iteratively calls ChIPMunk to check for several motifs possibly present in a given dataset.

What can I analyze with ChIPMunk?

ChIPMunk performs the best on ChIP-Seq or similar data (e.g., ChIP-exo) with the available "peak shape" profiles. However, any "alignable" data is OK, including HT-SELEX (SELEX-Seq), or footprinting data or ever sequences grabbed from protein-binding microarray. Shorter sequences generally produce more reliable alignment in the case the prior positional preference data is not available.

Mononucleotides vs dinucleotides

ChIPMunk comes in two: with classic commonly used mononucleotide motifs and improved dinucleotide motifs. The command-line syntax and output are almost the same (with the small differences specially mentioned below in the respective sections). Dinucleotide version is somewhat slower and requires more computational time to converge.

It is not difficult to select whether you should use ChIPMunk or diChIPMunk.

Mononucleotide version is to be used when: (a) you do not know anything about motifs in your data ("draft" run); (b) you plan to use other tools for downstream analysis (most of the existing tools will be able to utilize only mononucleotide matrices).

Dinucleotide version is better suited to produce a more precise representation of the optimal TFBS binding model. This would allow properly estimating the number of sequences containing motif hits. e.g. to measure the percentage of "the most reliable" ChIP-Seq peaks in a given dataset.

In terms of the consensus sequence, in general you should get very similar results from the mono- and dinucleotide versions.

ChIPMunk motif length estimation

ChIPMunk searches for the longest strong motif starting from the **`start_motif_length`**.

If **`start_motif_length`** > **`stop_motif_length`** then ChIPMunk decreases length by 1 at each step until strong motif is not reached.

If **`stop_motif_length`** > **`start_motif_length`** then ChIPMunk increases length by 1 at each step and stops when first weak motif is found (taking previously found strong motif).

If **`stop_motif_length`** = **`start_motif_length`** then ChIPMunk searches for the best motif of a given length.

Strong motif is defined by the discrete information content of its flanking columns. Mononucleotide version additionally checks for weak information content columns in the middle of the motif. Formal definition of the strong motif is given in a separate "math" section below.

Typically, it is reasonable to scan the lengths range from the shorter to longer lengths. If nothing is known about the motif few "draft" runs with a fixed length are suggested to roughly estimate the reasonable lengths range.

Command line parameters

Full command-line format is:

```
java -jar chipmunk.jar ru.autosome.ChIPMunk
  <start_motif_length> <stop_motif_length>
  <verbose>=(y)es|(n)o <mode>=oops|zoops_factor=0.0..1.0
  <x:>input_set1.mfa..<x:>input_setN.mfa
  <try_limit> <step_limit> <iter_limit> <thread_count>
  <seeds>=random|filename.mfa <gc%>=0.5|auto
  <motif_shape>=flat|single|double
  <disable_log_weighting>
```

The parameters are the same for diChIPMunk (with a few exceptions discussed below); ru.autosome.di.ChIPMunk package should be specified. The command-line parameters are given in <..> brackets. The parameters after the input_sets are optional and can be omitted.

Detailed description

```
<verbose>=(y)es|(n)o
```

Use **`y`** or **`yes`** for the additional details in the program output: ChIPMunk will provide the list of words used to construct the motif.

```
<mode>=oops|zoops_factor=0.0..1.0
```

`OOPS` or **`oops`** corresponds to the one-occurrence-per-sequence mode. This means ChIPMunk will produce a gapless multiple local alignment of sequences. This is OK for high-quality dataset without any noise. Typically you'll want to use zero-or-one-occurrence-per-sequence mode, where some sequences are expected to carry no motif occurrences. In this case, you should specify the zoops factor parameter, a value between **`0`** and **`1.0`**. The values closer to zero will result is a stricter threshold selection procedure raising the threshold and throwing more words of the alignment out. Recommended value: **`1.0`**.

```
<x:>input_set1.mfa <x:>input_set2.mfa .. <x:>input_setN.mfa
```

Note there is not space between type of the sequence set , **`x:`**, and the name of the multi-fasta file.

Here **`x:`** can be:

`s:` for simple multi-fasta to be searched in a double-strand DNA mode (the most common choice)

`r:` for simple multi-fasta that should be considered in a single-strand mode (e.g. for discovery of RNA motifs). Please note, that in this case the P-value estimation will not be fully correct (it is optimized for the 2 strand usage, see the respective section for details).

`w:` for weighted data set where the number in the fasta header specifies the sequence weight (quality, robustness, reliability, etc), i.e. specifies the impact this particular sequence has on the final motif. Toy example of the weighted data set with the second sequence two times "better" than the first:

```
> 1.0
ACGGGAAA
> 2.0
GTGAAAAA
```

`p:` peak data with the positional preferences profile. Each number in the space-separated list in the fasta header specifies the weight of the corresponding position. Toy example of the peak data set:

```
> 1.0 2.0 1.0 1.0
```

```
ACCG
> 2.0 10.0 1.0 1.0 2.0
GTACA
```

Real life examples can be found in the sequence data sets of the HOCOMOCO collection, see, e.g. AP2A ChIP-Seq data at http://autosome.ru/HOCOMOCO/modelDetails.php?tf=AP2A&model=f2

The peak positional profile is useful for any sequence-specific positional prior.

In a single run ChIPMunk can integrate data from different sequence sets of different types (so you can use peak and weighted and simple data sets together). In particular, this is useful if you have a very noisy dataset but know a putative consensus of a motif. In this case, you should make a fasta-file with the consensus and supply it as one of the input data sets. See the description of the `disable_log_weighting` parameter below for more information.

`<try_limit>`

Default value: **`100`** for ChIPMunk, **`200`** for diChIPMunk. This is an internal number of motif optimization runs. For a random seeding, this would be simply equal to the number of seeds. It can be as high as your computational power (100-1000 seems to be generally enough depending on your dataset). In practice for good sequence sets and strong motifs the value of **`100`** is acceptable. The value should be increased in case of the unstable ChIPMunk behavior on weak motifs or noisy datasets. For the final run, when you know the proper motif length, the values can be increased (upto 200 and 400 respectively).

`<step_limit>`

Default value: **`10`** for ChIPMunk, **`20`** for diChIPMunk. Corresponds to the number of subsampling/bootstrapping cycles used for each seeded optimization (see above). Can be increased for small/noisy datasets and weak motifs.

Both **`step_limit`** and **`try_limit`** increase computational time in a linear fashion.

`<iter_limit>`

Default value: **`1`** (recommended fixed value). This is a number of optimization cycles ChIPMunk does on a subsample of sequences. Higher values will reduce speedup gained by subsampling and will have no significant effect on motif quality. Parameter is included in the list to maintain command-line compatibility with the previous ChIPMunk version.

`<thread_count>`

Default value: **2** (2 to 4 are recommended). This is a size of the computational thread pool used by ChIPMunk. Parallel nature of the ChIPMunk algorithm allows improving computation time by usage of multiple CPU cores. Higher values will require more memory and generally would not lead to noticeable speed improvement.

```
<seeds>
```

Can be either **`random`** (default, generates random seeds for PWM optimization) or can specify the name of the multi-fasta file to rip seeds from given sequences.

```
<gc%>
```

This parameter specifies the GC percent of the background nucleotide composition of the sequence sets. Default value: **`0.5`** (uniform) for ChIPMunk. For diChIPMunk the background dinucleotide distribution is taken uniform by default. When **`auto`** or **`local`** is specified (di)ChIPMunk will compute the background nucleotide composition from supplied sequence sets.

```
<motif_shape>
```

This parameter is **`flat`** (no preference) by default. It also can be **`single`** and **`double`** corresponding to the single-box and double-box motif shape priors with the information content optimized to fit DNA helix pitch. I.e. the positions within motif are weighted, so the positions with higher weight have more impact on the PWM score during optimization. The single box prior uses $\cos^2$(PI*n/T) weighting, the double box prior uses $\sin^2$(PI*n/T) weighting. Here PI = 3.1415926, T is the DNA helix period (10.5bp), and n is the position within the motif.

```
<disable_log_weighting>
```

This parameter can be set to any value to disable the logarithmic data set weighting strategy. In

v3-v4 ChIPMunk assigned the dataset weight proportional to the natural logarithm of the number of sequences in the set. The weights of individual peaks of the peak datasets were also logarithmically scaled according to the maximal peak height of each peak.

Setting this parameter to any value will allow using plain weighting strategy when the datasets have equal impact on the motif independently of the number of sequences in each dataset. This is useful when you add a fake dataset containing only a single sequence with selected consensus to be aligned together with your sequences to guide motif discovery. Also this parameter disables logarithmic weighting of peak heights for peak datasets. In turn, this may be useful to give preference to higher peaks. Still, this may be dangerous in analysis of ChIP-Seq data, since few highest peaks can be artifacts (caused either by PCR or by wrong read alignment) and their extreme coverage will add too much noise. In the default logarithmic weighting scheme the impact of such peaks is not so dramatical so no special processing is necessary.

Examples

Motif lengths from 7 to 10; one standard sequence set data.mfa; one-occurrence-per-sequence; verbose mode on; default number of seeds/subsampling iterations:

```
java -cp chipmunk.jar ru.autosome.ChIPMunk 7 10 yes oops
s:data.mfa
```

User-specified values for seeds/subsampling iterations; motif lengths decreasing from 20 to 7; two sequence sets with different data models (peak and weighted); zero-or-one-occurrence-per-sequence mode; verbose off:

```
java -cp chipmunk.jar ru.autosome.ChIPMunk 7 20 no 1.0
p:chipseq.fasta w:selex.mfa 1000 100 1
```

Peak data used; user-specified values for seeds/subsampling iterations; motif lengths from 10 to 20; verbose on; strict threshold selection in zero-or-one-occurrence-per-sequence mode; 4 computational threads; random seeds; predefined gc% 0.6:

```
java ru.autosome.ChIPMunk 10 20 yes 0.0 p:data.fasta 10000 100 1 4
random 0.6
```

Preparing your data

Most of the sequence sets are already in multi-fasta format. Yet, to get the most of the ChIPMunk, you'll need to prepare peak profiles (when possible); this is especially important for ChIP-Seq data. To this end you'll need to parse 'bedGraph' or 'wiggle' files and place the positional profile along to the fasta headers of the respective sequences. More information on data formats is given at UCSC, see http://genome.ucsc.edu/FAQ/FAQformat.html. Real-life examples of the peak profile-equipped sequence sets can be found on the ChIPMunk website.

Generally, peak profiles will significantly improve the results of the motif discovery and even allow you to be sure that no significant peak-associated motif is found. When no positional preference data is given you may find strong but non-relevant motif, e.g., somewhere far from the peak summits and this will require additional processing to see whether the motif actually lies under the summits or not.

Practical tips

There are few ideas, which will help you to get the most of ChIPMunk.

0. Don't hesitate to start from basic ChIPMunk runs with fixed lengths (10-15-20). This will help you to check whether the motif is really there in you data, and which lengths range is reasonable. At this step you may even reduce the default number of iterations/seeds to get a quick-and-dirty look on your data.

1. If you have ChIP-Seq base coverage profiles (i.e. peak shapes) then you should use the peak mode and uniform background.

2. If you have ChIP-Seq peaks without base coverage data but with peak summit locations then you should generate a triangle positional profile (with the top value of 1.0 at the peak summit and values linearly decreasing from 1.0 at peak summit to zero at peak borders).

3. If you have any known positional preferences then you should produce a positional profile and use the peak mode.

4. If your motif is weakly defined or has strange-looking consensus then you should try motif shape priors (single-box and double-box, optimized for the one and to DNA helix turns).

5. If your sequences have different reliability then you should assign some empirical weights to them and use the weighted sequences mode.

In general, positional profile provides the best results, and it is safe to use it with the uniform background assumption (default mode). In the case of an unknown positional profile, it is advised to use auto-detected local background (auto). Motif shape priors may help in both cases.

If you are getting different random garbage from different ChIPMunk runs then either your positional prior are wrong, or the sequence set contains too much noise. If your data is noisy, or the motif is weak, or the sequences are too long, or the results are not stable: try manually selecting reasonable fixed motif length and then increase the number of iterations/seeds. In some cases, this would allow to detect a more stable signal.

Understanding the results

ChIPMunk and ChIPHorde print the running status messages into the standard error stream, all the results go to the standard output stream.

ChIPMunk iteratively calls ChIPAct (a fixed length motif search); and ChIPHorde iteratively calls ChIPMunk. Thus, the output consists of several sections, surrounded by `OUTC|` and `DUMP|`:

```
OUTC|ru.autosome.ChIPAct

DIAG|gapless multiple local alignment of length 10

KDIC|0.5752588869008193

TIME|14.721

DUMP|ru.autosome.ChIPAct V6 17052014
```

each of which corresponds to the separate run of the ChIPAct (for ChIPMunk) or ChIPMunk (for ChIPHorde) module.

Generally, the final result of the program run, i.e. the resulting motif, **is located in the very end** of the results file.

The output format is quite straightforward with each line being a “tag|value” pair, where the “tag” describes the content presented in the “value” which is separated by “|”.

The tags `OUTC` and `DUMP` mark the beginning and the end of the output of a particular module execution. `DIAG` shows diagnostic informative messages, `KDIC` is the Kullback Discrete Information Content of the motif (see the Math section below for details) and `TIME` is the execution time in seconds.

The complete list of tags of the ChIPMunk output

The ChIPMunk results section starts from the `OUTC|ru.autosome.ChIPMunk` and ends with `DUMP|ru.autosome.ChIPMunk`.

`TOTL` – the total number of sequences used in the run (discarding too short sequences)

`WRDS` – the number of words included in the motif. Can be higher that the number of used sequences, since several equally scoring words can contribute to the motif.

`SEQS` – the number of sequences included in the motif (in one-occurrence-per sequence mode it is always the same as TOTL).

`COVR` – coverage of the input sequence set obtained by the motif (`SEQS` / `TOTL`); in one-occurrence-per-sequence mode it is always 1.0.

`LIST` – the header of the list of words included in the motif. Printed only in the verbose mode. Followed by the `WORD` records for each word. The weight column shows the weights of the words, i.e. how much impact they had on the motif (depending on the sequence-weighting scheme).

`LENG` – the detected length of the motif. Can be -1 if the length was fixed (the range was from *X* to *X*), or if the lengths detection failed (see `DIAG` in this case).

`KDIC` – Kullback Discrete Information Concent (`KDDC` for diChIPMunk).

`A C G T` – show the weighted positional counts of each nucleotide letter. These values can be safely threated as counts in the most cases or (if this is strictly necessary for some applications) rounded to the nearest integer value. `AA AA AC` .. `TT` tags are present for diChIPMunk.

`N` – the total weight of the alignment (the whole set of the words included in the motif). In case there was no weighting scheme, N is equal to the number of words.

`PWMA PWMC PWMG PWMT` – the log-odds transformed counts of the production-ready positional weight matrix. `PWAA PWAC` .. `PWTT` are present for diChIPMunk.

`THRE` – the score threshold value for the respective weight matrix.

`IUPA` – IUPAC consensus sequence corresponding to the respective count matrix.

`PVAL` – **<u>(experimental)</u>** the Binomial P-value estimated for the motif, the threshold and the input sequence set.

`BACK` – background nucleotide distribution (dinucleotide for diChIPMunk) used for motif discovery (either auto-estimated from the data, uniform or predefined by the user).

`ERRR` – error during program execution, see the indicative message.

Additional ChIPHorde tags

ChIPHorde output starts from the `OUTC|ru.autosome.ChIPHorde` and ends with the `DUMP|ru.autosome.ChIPHorde`.

There are several specific tags.

`INFO` – informational message on the number of discovered motifs.

`MOTF` – the index of the motif record appearing below.

`OCCS` – the occurrence line appearing in the verbose mode. In this case, for each detected motif ChIPHorde additionally scans all the input sequences gathering the motif hits passing the threshold.

The line looks like

```
OOCS|0;44; ACCCCTCCCACT:5:DIRECT:7.95615282898139
```

where `0` is the number of dataset in the input data (always 0 if you supply only one fasta-file), `44` is the number of sequence in the set, `ACCCCTCCCACT` is the detected word, `5` is the position of the word in the sequence, `DIRECT` is the strand of the occurrence. The PWM score is given in the end.

All indices are ZERO-based.

Interpreting the results

How you can be sure your motif is OK?

First, run ChIPMunk two or three times and check whether results are similar. ChIPMunk is stochastic algorithm, but if there is a robust signal in the data then it should be able to dig it stably.

Second, check KDIC of the detected motif. KDIC of ~0.5 is generally OK (more is better, but such level of sequence conservation is unlikely to be detected in real data expect for special cases).

Third, check <u>(experimental feature)</u> the P-value of the detected motif. Small P-values (<< 0.05) are likely significant (i.e. they confirm the detected motif is not garbage). Large P-values (> 0.05) are non-significant.

The P-value measure by ChIPMunk is the probability to obtain a given number of sequences with at least one motif hit above the selected threshold in each. For longer sequences P-value will be less significant (since only the single best motif occurrence per sequence is taken into account). For shorter sequences P-value will be more significant even in case of weaker motif and thresholds.

Please note, that ChIPMunk uses the "self-consistency" test to select the threshold for the produced PWM. P-value is estimated for this auto-detected threshold. Technically, if the motif length was wrongly estimated, or if the sequence set contained too many weak binding sites, the threshold can be too weak to produce significant P-value.

Thus, non-significant P-value does not mean the motif is bad; but it suggest you to rerun ChIPMunk using different motif lengths range; or even retry the same motif lengths range (since the algorithm is stochastic it may lead detect a slightly different motif with a slightly different and more robust threshold).

Finally, check other motif collections (e.g. http://autosome.ru/HOCOMOCO/ for human TFs) for motifs similar to that found in your run. For example, you can use http://opera.autosome.ru/macroape/.

Looking for several motifs by (di)ChIPHorde

(di)ChIPMunk can arrive in a horde. (di)ChIPHorde allows iteratively look for different motifs in a given dataset using either masking (replacing motif hits with poly-N-s) or filtering (removal of sequences with motif hits).

Since ChIPMunk is stochastic, each consequent round of filtering becomes less and less stable so the ChIPHorde results may significantly change from run to run. Thus, it is advised (a) to use "draft" runs with fixed motif lengths ranges to estimate which motifs are there in the data and (b) to run ChIPHorde several times and collect all reasonable results.

Please note that major motif is generally OK to use as the model, i.e. to use it to predict motif occurrences in different data sets. Secondary motifs are much less robust and typically have significantly lower quality (since they are lesser represented in the input data).

Examples and details

Full ChIPHorde command-line format is:

```
java -jar chipmunk.jar
<start_length1:stop_length1,start_length2:stop_length2..start_leng
thN:stop_lengthN> <mode=(m)ask|(f)ilter|(d)ummy>
<verbose=(n)o|(p)artial|(y)es> <ru.autosome.di.ChIPMunk
parameters>
```

ChIPHorde uses ChIPMunk command-line format with a few modifications. For example

```
java -cp chipmunk.jar ru.autosome.ChIPHorde 8:10,12:6 mask yes 1.0
s:BCD_footprint.mfa > test_result.out
```

sequentially runs ChIPMunk twice using **`8:10`** and **`12:6`** motif lengths ranges searching for the first motif starting from the shortest possible length and for the second motif from the longest possible length; **`mask`** means to sequentially mask (poly-N) occurrences of each detected motif before the next round of the motif discovery.

There is another option, **`filter`**, which will make ChIPHorde to exclude whole sequences containing proper motifs hits. Another option, **`dummy`**, can be used to check different length ranges using ChIPHorde without any filtering, i.e. this simply allows to run ChIPMunk several times so many 'versions' of the same motif can be obtained:

```
java -cp chipmunk.jar ru.autosome.ChIPHorde 10:10,15:15,20:20
dummy yes 1.0 s:sequences.mfa > test_result.out
```

Modes **`mask`** and **`filter`** are suitable for different conditions. Typically mask are to be used searching for co-factor motifs while filter is intended for searching for (possibly) different motifs of a single TF or to filter some 'motif contamination' (e.g. sequences enriched with satellite repeats) before searching for the 'true' motif.

ChIPHorde uses ChIPMunk-defined threshold for masking/filtering, so ChIPHorde will work only with the specified zoops factor, i.e. when ChIPMunk is executed in zero-or-one-occurrence-per-sequence mode.

If you want more motifs, the zoops factor is better to be set to zero to ensure that at each step the worst selected word still has good PWM score (and thus the threshold wouldn't be too low). To get 2 top motifs it's OK to use default 1.0 zoops factor.

The ChIPHorde command-line is very similar to that of ChIPMunk: you can specify the iteration counts, specific GC-content to account for and so on.

For the **`verbose`** setting there is **`partial`** (or **`p`**) mode when only aligned words are printed, and the full verbose mode (**`yes`**) when, additionally, all motif occurrences are printed.

ChIPHorde motif occurrences report

In case the full verbose mode was specified ChIPHorde gathers all occurrences of detected motifs in the whole set of sequences, the respective lines are marked with the **`OCCS|`** key in the ChIPHorde output.

A proper line looks like

```
OCCS|0;44; CGCCTAATCT:6:DIRECT
```

where **`0`** is the number of dataset in the input data (always **`0`** if you supply only one input fasta-file),

44 is the number of sequence in the set, **CGCCTAATCT** is the word at that position, **6** is the index of the word in the sequence, **DIRECT** is the strand orientation of the occurrence.

Please note, that all indices are ZERO-based. What is no less important, if too short sequences are filtered during sequence loading stage, and the reported indices of the sequences account only for properly loaded sequences.

Supporting tools

A set of supporting tools for ChIPMunk includes (a) output parser & motif LOGO plotting scripts (pmflogo) in ruby (uses RMagick/ImageMagick); java-version is under development and (b) java-based scanning tool to find motif occurrences in a given set of sequences using (di)ChIPMunk (di)PWMs.

SPRY-SARUS

SPRy-SARUS stands for Straightforward yet Powerful Rapid SuperAlphabet Representation Utilized for motif Search. This is a simple tool which uses superalphabet approach presented by [Pizzi, Rastas & Ukkonen; 2007] to scan a given set of sequences for (di)PWM hits scoring no less than a given threshold. SPRY-SARUS uses fairly simple command line format, accepts weight matrices in plain text files (with log-odds or similarly transformed weights) and DNA sequences in multifasta. SPRY-SARUS is written in Java and. The all-in-one jar files are available on the ChIPMunk web page. The proper command line format is printed if SARUS is executed w/o arguments:

```
java -cp sarus.jar ru.autosome.SARUS
```

or in a shorter form

```
java -jar sarus.jar
```

The output from this command is:

```
SPRY-SARUS command line: <sequences.multifasta> <weight.matrix>
<threshold>|besthit [naive] [suppress] [transpose]
```

The arguments have self-speaking names. The weight matrix can be given either with or without header line (starting from “>”). By default each line in the file corresponds to a single position of a motif, i.e. each line should contain 4 (or 16) elements for PWM (diPWM).

Please note, that SARUS can use raw ChIPMunk (but not ChIPHorde) output extracting the resulting motif right from the log-file (if the ChIPMunk output was redirected into a file as suggested in this guide).

The “transpose” modifier suggests SARUS to use the transposed file format; “suppress” modifier suppresses sequences names in output; “besthit” modifier can be used instead of a threshold value to force SARUS look for a single best hit in each sequence; “naive” modifier switches from the superalphabet to a naive scanning mode and is rarely useful.

The output format is fairly simple showing the sequence header (via “>” as in the input multifasta file), the position and the strand orientation of a PWM hit (is passing the threshold).

The dinucleotide version can be used in a similar way:

```
        java -cp sarus.jar ru.autosome.di.SARUS
```

As for ChIPMunk, it is convenient to redirect SARUS output to a file using command line pipe redirection. By default all results are printed to standard output stream.

MACRO-APE and PERFECTOS-APE

MAtrix CompaRisOn and PrEdicting Regulatory Functional EffeCT of SNVs by Approximate P-value Estimation software suite is useful to estimate thresholds, motif P-values, detect motifs similar to a given query and scan SNPs or SNPs for overlapping and changing TFBS. Please visit autosome.ru “Opera House” for a web version and more information on these tools: http://opera.autosome.ru

ChIPMunk-on-the-web

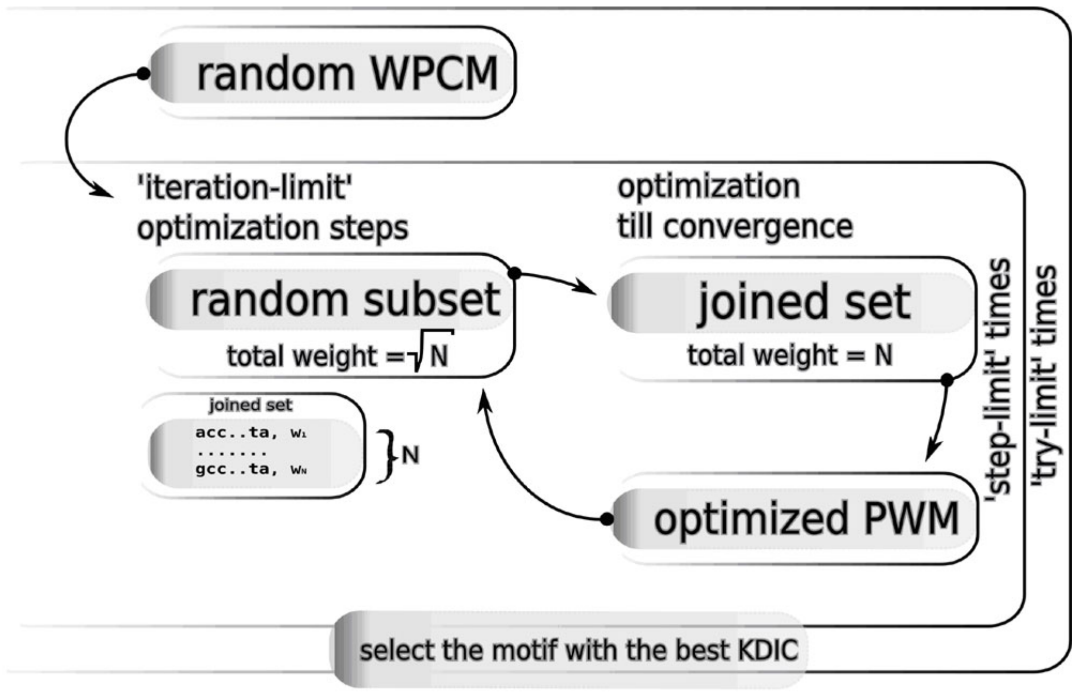
Currently only old (version 3) ChIPMunk is available through the web interface. The actual version 6 covered by this guide is prepared to be available as online tool. Up-to-date stand-alone version is already provided.

Formal math

Algorithm flowchart and pseudocode

Original flowchart of ChIPMunk core

Here N is the total number of sequences in the all input sets. ChIPMunk makes 'step-limit' subsampling iterations for each seeding (starting) WPCM (for each of 'try-limit' different seeds).



ChIPMunk "best-GMLA-best-PWM" core pseudocode representation

```
define KDIC_COLUMN_MAX = -Log(0.25)
define the_best_matrix

try_limit times do

  define matrix = make_random_matrix
  define current_best_matrix

  step_limit times do

    subset = subsample(joined_set)
    optimize(matrix, subset) iteration_limit times
    optimize(matrix, joined_set) until convergence
    if KDIC(matrix) > KDIC(current_best_matrix) then
       current_best_matrix = matrix
    if KDIC(matrix) + KDIC_COLUMN_MAX < KDIC(the_best_matrix) then
       matrix = the_best_matrix

  end

  if KDIC(current_best_matrix) > KDIC(the_best_matrix) then
     the_best_matrix = current_best_matrix

end
```

Here `KDIC_COLUMN_MAX` is the maximal attainable KDIC of a single GMLA column. See below for a KDIC definition. This constant was introduced in newer ChIPMunk versions to substitute current solutions for the existing "better" one only if a current solution was significantly worse. This ensures that an obtained sub-optimal solution is not automatically discarded in a favor of existing local optima, but has a chance to be improved in further subsampling runs leading to a different, possibly better local optima than that already found.

GMLA is constructed from the best hits of a respective PWM. diChIPMunk uses the same procedure but diPWMs for motif representation and KDIDIC as PWM-alignment quality control.

Positional weight matrix

PWM is a simple and the most popular way to represent text patterns in biological sequences. Typically, PWM construction starts from a gapless multiple local alignment of sequences, containing occurrences of a studying pattern. By counting "letters" in each column of the alignment one constructs a positional count matrix (PCM), a matrix of 4xL integer values representing frequencies of letters observed in each alignment column, where L is the length of the studying pattern. ChIPMunk can utilize a weighting strategy for input sequences, with higher weights to be assigned to more reliable sequences (using some experimentally justified quality values). In this case, the letter counts are also going to be "weighted" and thus the PCM will contain real numbers (still reflecting the letter frequencies, but with weighting coefficients applied).

The PCM (or weighted PCM, WPCM) of "counts" or "frequencies" can be converted to PWM with additive weights using any common transformation, such as logarithmic transformation with background normalization, also known as log-odds. A pseudocount value is typically introduced to PCM to avoid zero frequencies, which otherwise prevent direct log-odds transformation. The exact formula for ChIPMunk WPCM-to-PWM transition is:

$$S_{\alpha,j} = log\left(\frac{x_{\alpha,j} + cq_{\alpha}}{(N+c)q_{\alpha}}\right),$$

where $S_{\alpha,j}$ is the score of letter $\alpha \in \{A,CAG, .., T\}$ in position j, $x_{i,j}$ is the number of occurrences of dinucleotide letter α in the j-th column of the GMLA, q_{α} is the background frequency of a nucleotide α, N is the total number of sequences in the GMLA, and c is the pseudocount parameter set as **natural logarithm** of N.

KDIC concept

The following KDIC description is taken exactly from the original ChIPMunk supplementary. The improved KDIC scaling is discussed in the next subsection.

Let' denote the distribution for background probability of nucleotide occurrence as $Q = \{q_1, ..., q_k\}$, where $\{1,..,k\} = \{A, C, G, T\}$. Consider a column j in an alignment of N sequences; the numbers of occurrences of nucleotides (letters) in this column are equal to $\{x_1, ..., x_k\} : \sum_{i=1}^{k} x_k = N$.

The probability to obtain the given column j in the *i.i.d* series of N tests given that the background probabilities of letter occurrences equal $Q = \{q_1, ..., q_k\}$ is

$$P(n_1, ..., n_k \mid j, Q) = \frac{N!}{x_1! ... . x_k!} q_1^{x_1} ... q_k^{x_k} . \quad (1)$$

Taking the logarithm:

$$\log P\left(n_1,\dots,n_k \mid j,Q\right)=\log\frac{N!}{x_1!\dots.x_k!}+\sum_{i=1}^{k}x_i\log q_i \qquad (2)$$

In our paper [Kulakovskiy, I.V., Favorov, A.V. and Makeev, V.J. (2009) Motif discovery and motif finding from genome-mapped DNase footprint data. *Bioinformatics*, **25**(18), 2318-2325] we have introduced discrete information content (DIC) as a measure of homogeneity of the nucleotide composition in a column j :

$$DIC(j)=\frac{1}{N}\left(\sum_{i=\{A,C,G,T\}}\log x_i!-\log N!\right). \qquad (3)$$

The overall number of possible different columns in the alignment of *N* sequences equals

$$I_N=\frac{N!}{x_1!\dots x_k!}=e^{-N\times DIC}. \qquad (4)$$

For large x_i, this value value can be approximated using Schneider's column specific information content [Schneider, T.D., Stormo, G.D., Gold, L. and Ehrenfeucht, A. (1986) Information content of binding sites on nucleotide sequences. *J Mol Biol*, 188, 415-431]:

$I_N=2^{N(2-I_S)}$, where $I_S=2+\sum_{i=\{A,C,G,T\}}p_i\log_2 p_i$. (5)

DIC does not take any background model into account.

Taking formulae (2) and (3) we obtain:

$$\log P\left(x_1,\dots,x_k \mid j,Q\right)=-N\cdot DIC+\sum_{i=\{A,C,G,T\}}x_i\log q_i \qquad (6)$$

The first term in (6) reflects the non-homogeneity of column *j* by itself, whereas the second term measures the divergence of the column *j* from the background. To understand deeper the meaning of (6) let us consider the approximation (2) for large x_i. In this case the one can use the Stirling's formula

$$\log n!\sim\frac{1}{2}\log\left(2\pi n\right)+n\log n-n\log e, \qquad (7)$$

and (2) is approximated as:

$$\log P(x_1,\dots,x_k \mid j,Q) \sim -\frac{(k-1)}{2}\log 2\pi + \frac{1}{2}\log\frac{N}{x_1,\dots,x_k} + \sum_{i=1}^{k} x_i \log\frac{N}{x_i} + \sum_{i=1}^{k} x_i \log q_i .$$

Keeping only the terms that are proportional to N and introducing column-specific frequency estimators $\tilde{p}_i = \frac{x_i}{N}$, we obtain:

$$\log P(x_1,\dots,x_k \mid j,Q) \sim -N \sum_{i=\{A,C,G,T\}} \tilde{p}_i \log \tilde{p}_i + N \sum_{i=\{A,C,G,T\}} \tilde{p}_i \log q_i = -N \sum_{i=\{A,C,G,T\}} \tilde{p}_i \log\frac{\tilde{p}_i}{q_i} \qquad (8)$$

For large x_i frequencies $\tilde{p}_i$ converge to the column-specific probabilities p_i of symbols: $P = \{p_i\}$; $i \in \{A,C,G,T\}$, and (8) can be rewritten using Kullback-Leibler divergence of *Q* from *P*, which reflects how difficult is to discriminate two populations taken from distributions *Q* and *P* with the best test imaginable [Kullback, S., and Leibler, R.A. (1951) On information and sufficiency, *Ann. Math. Statist.*, **22**(1), 79-86]:

$$D_{KL}(P \mid Q) = \sum_{i=\{A.C.G.T\}} p_i \log\frac{p_i}{q_i} \qquad (9)$$

$$\log P(p_1,\dots,p_k,N \mid j,Q) \sim -N D_{KL}(P \mid Q) \qquad (10)$$

Thus, we can rewrite (6) as

$$\log P(x_1,\dots,x_k \mid j,Q) = -N \cdot KDIC(j) \qquad (11)$$

where

$$KDIC(j) = \left(DIC(j) - \sum_{i=\{A,C,G,T\}} \frac{x_i}{N} \log q_i \right) \qquad (12)$$

is the discrete analog of Kullback-Leibler divergence.

Note that DIC is always negative with the maximum of zero. One the other hand the subtracted term is always negative too (because $q_i \le 1$). Thus, KDIC can be either positive or negative.

In the case of a binding motif of the length L, the measure of the quality and of the divergence from the background is obtained by accumulation of KDIC of all columns:

$$KDIC = \sum_{j=1}^{L} KDIC(j) \qquad (13)$$

This value has been used as an optimization criterion for the best motif.

It should be noted, that counts $\{x_1, \ldots, x_k\}$ in the case of weighted counting can be real numbers. Actually, to compute factorials we used the Stieltjes approximation [Abramowitz, M. and Stegun, I.A. (1972) Handbook of Mathematical Functions. *Dover Publications*, New York], which also holds for real values. This approximation was used in our computations both for integer and for real values.

Improved KDIC scaling

What is not very convenient with the original KDIC definition is that its values depended both on the motif lengths (longer motifs have higher KDIC values) and on the number of aligned words `N`. The motif length normalization is quite easy (dividing the final values by the motif length). Thus the remaining step is to scale KDIC of a particular column. To this end one needs estimating the attainable KDIC min-max values for a given `N`. The maximal column KDIC value is obtained if the most rare letter (with the lowest background probability `q`) is 100% conserved and all other letters do not occur at all:

```
KDIC_COLUMN_MAX = -log(q)
```

For simplicity ChIPMunk uses the uniform background distribution for KDIC scaling, i.e. `q` = 0.25. There is a possibility that KDIC values will exceed 1.0 in the case of strong motifs in a very non-uniform background, but practically this should be a very rare event.

The minimal KDIC can be estimated having in mind that DIC (see above) can be approximated with $\sum_\alpha p_\alpha \log p_\alpha$ and thus KDIC=$\sum_\alpha p_\alpha \log \frac{p_\alpha}{q_\alpha}$, where `p` is the normalized observed frequency (~probability) of a particular letter `α` in a motif column and `q` is the background probability. The worst worth-considering motif has letter frequencies fully consistent with the background distribution. Thus, the minimal KDIC of a motif column should be **zero**. Theoretically, negative KDIC values may exist (if the studying motif for an unknown reason notably favors more frequent letters of a background distribution). However, again, this should be never encountered in real-life applications.

KDIC formula

The final KDIC formula as used in the last ChIPMunk is:

$$\text{KDIC} = -\frac{1}{l \log q} \sum_{j=1}^{l} \frac{1}{N} \left(\sum\nolimits_{\alpha \in \{A,C,G,T\}} \log x_{\alpha,j}! - x_{\alpha,j} \log q_\alpha - \log N! \right)$$

where `l` is the motif length, `q` is a background frequency of a single nucleotide (`q=0.25` in the case of the uniform nucleotide distribution), `N` is the total number of words in the GMLA, `α` is the nucleotide letter, $x_{\alpha,j}$ is the raw non-normalized frequency (count) of a particular nucleotide `α` in a counts matrix and GMLA column `j`. KDIC values mostly fall into `[0,1]` range with those `~0.5` being considered good enough.

Dinucleotide weight matrices and KDIDIC concept

Dinucleotide extension of ChIPMunk is based on the dinucleotide superalphabet of 16 dinucleotides. This allows directly accounting for interposition dependencies by representing the sequences in dinucleotide alphabet with each letter "AA","AC",.."TT" depending on its neighbors. For example, ACGT nucleotide sequence is written as A-C-G-T in nucleotides and as AC-CG-GT in dinucleotides. This allows (a) using the same weight matrix representation but on a wider alphabet (i.e. the interposition "dependencies" are taken into account implicitly) and (b) using KDIC almost "as is" but with a different alphabet. We call this measure as Kullback Dinucleotide Discrete Information Content, KDIDIC and define it as follows:

$$\mathrm{KDIDIC} = -\frac{1}{l \log q} \sum_{j=1}^{l} \frac{1}{N} \left(\sum\nolimits_{\alpha \in \{AA,AC,AG,..TT\}} \log x_{\alpha,j}! - x_{\alpha,j} \log q_{\alpha} - \log N! \right)$$

where `l` is the motif length (in dinucleotides), `q` is a background frequency of a single dinucleotide (`q=0.0625` in the case of the uniform nucleotide distribution), `N` is the total number of words in the GMLA, `α` is the dinucleotide letter, $x_{\alpha,j}$ is the raw non-normalized frequency (count) of a particular dinucleotide `α` in a counts matrix and GMLA column `j`. Similarly to KDIC, KDIDIC values mostly fall into `[0,1]` range with those `~0.5` being considered good enough.

KDIDIC does not explicitly account for dependencies of neighboring dinucleotide columns, however, dinucleotide sequences are unambiguously mapped to corresponding mononucleotide sequences written in 4-letter ACGT-letter alphabet. Thus, neighboring positions (e.g. `j` and `j+1`) of the alignment are actually dependent since the neighboring dinucleotide letters overlap in the initial DNA nucleotide sequence.

ZOOPS mode mechanics

ChIPMunk searches for the gapless multiple local alignment that has the maximum KDIC value. Then a positional weight matrix (PWM) is constructed, and all the aligned words are sorted by their PWM scores. They are classified into "the signal" and "the noise" sublists. To find the boundary word we construct a series of PWMs, made from top `1,2,…,n` words. The idea is that for "the signal" the score of `n`-th word calculated with the `n`-th PWM should be notably greater than that calculated with `N`-th PWM (where `N` is the total words count).

To formalize this idea we maximize the following sum:

$$\max_{n=1..N} \sum_{i=1}^{n} \left(score\left(\mathrm{PWM}_n, \mathrm{word}_i\right) - score\left(\mathrm{PWM}_N, \mathrm{word}_i\right) \right)$$

Here word_i is the aligned word in the sequence `i`.

PWM scores depend on the number of words used for its construction according to the formula:

$$S_{\alpha,j} = log\left(\frac{x_{\alpha,j} + aq_\alpha}{(c+a)q_\alpha}\right)$$

where $S_{\alpha,j}$ is the score of letter $\alpha \in \{A,C,G,T\}$ in position j, $x_{\alpha,j}$ is the number of occurrences of letter α in the j-th column of the word list, q_α is the background frequency of nucleotide α, c is the total number of words in the list, and a is the pseudocount parameter.

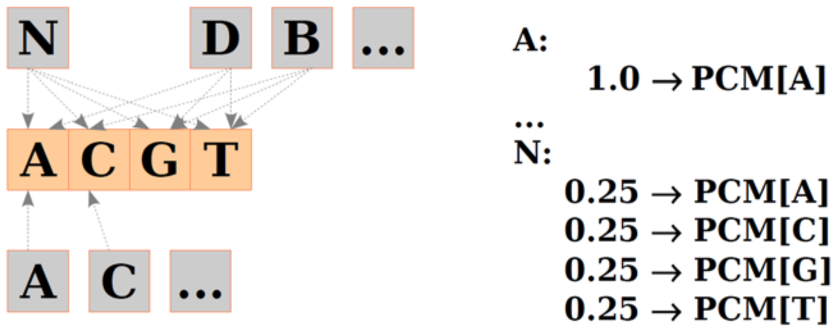
Since N-th and n-th PWMs are constructed from the different number of words, and PWM scores depend on the number of source data, one need to rescale PWMs before comparison. To this end, we have adjusted the pseudocount values to make the minimal and the maximal PWM scores identical. This can be achieved by setting pseudocount values proportional to the words count, i.e. $a = kc$. Larger pseudocounts correspond to the more flexible segmentation (i.e. more word included into the motif, i.e. less stringent thresholds). The value of $k=1.0$ is suggested for default practical usage.

Accounting for positional prior and weighted sequences

One of the important ChIPMunk features is its ability to handle positional profiles along the input sequences as prior information on motif positional preferences.

The idea came from processing of IUPAC symbols representing ambiguous positions with several possible nucleotides, with an extreme case of unknown letter "**N**" denoting all 4 letters possible in a particular position (not to be confused with the number N denoting the total number of words).

A basic scheme for IUPAC letters and weighted sequences



At the PCM construction stage, each IUPAC letter count is proportionally split between "normal" letters. At the PCM aggregation stage (not shown) the IUPAC letter counts are recalculated from the basic letter counts (averaging the counts of the respective letters, e.g. averaging **G** ant **T** counts for IUPAC **D**).

PCM construction scheme for weighted sequences and positional profiles

weighted sequence (w_i)	**A:** ... $w_i \rightarrow \mathbf{PCM[A]}$
weighted sequence with the positional profile	**A:** ... $\text{profile}_j w_i \rightarrow \mathbf{PCM[A]}$

In formal math:

$$\alpha \in \{A,C,G,T\}: \quad x_{\alpha,j} = \sum_i w_i \cdot \text{profile}_i[j] \cdot \delta(\alpha, \text{word}_i[j])$$

$$\alpha = \mathbf{N}: \quad x_{\mathrm{N},j} = \sum_i w_i \cdot (1 - \text{profile}_i[j])$$

$$\forall j: \sum_{\alpha \in \{A,C,G,T,\mathrm{N}\}} x_{\alpha,j} = \sum_i w_i \qquad \delta(a,b) = \begin{cases} \text{if } a = b \text{ then } 1 \\ \text{if } a \neq b \text{ then } 0 \end{cases}$$

Here i denotes the index of the sequence in the input set, j denotes a particular sequence position, and `profile` is the vector of positional information (scaled to [0..1] range with its height saved as sequence weight w).

PWM scores for sequences with weight profiles

The PWM scores for sequences with position profiles are computed quite easily (note that the sequence weight is not used here since PWM scores are used within a single sequence to select the best hit):

$$score(\text{PWM}, \text{word}) = \sum_{j=1}^{length} S_{\text{word}[j],j} \cdot \text{profile}[j]$$

Estimating motif P-value (experimental)

Usage of KDIC of a GMLA is OK when comparing motifs in OOPS mode. When it comes to ZOOPS mode, strict segmentation may produce motifs with high KDIC values when a score threshold is high and only a small subset of the word list is included into the motif.

To properly assess whether the discovered motif worth considering we introduced the motif P-value which is estimated as a probability to find a given number of sequences each having at least one motif hit passing ZOOPS-selected threshold. The null hypothesis is that the sequences in the total input set were generated by a random model (“Bernoulli” for ChIPMunk and Markov(1) for diChIPMunk). The P-value is estimated using Binomial distribution with a number of trials equal to the number of sequences in the input set, the number of successes is the number of sequences with motif hits, the success chance p is the probability for a random sequence of a fixed length to contain a motif hit scoring no less than the threshold.

The success chance `p` is estimated using MACRO-APE to get the probability p_0 for a random word to score no less than the threshold; and then p is calculated under the assumption that motif hits are independent of each other: $p = (1-p_0)^{2(L-l+1)}$.

`L` is selected as the median length of sequences in the input set and `l` is the motif length and `2` comes from the fact that we consider both strands (thus the approximation will not hold for single-stranded “RNA” mode).

Licensing policy

ChIPMunk is distributed under the **WTFPL** license. Yet, we would be happy if you cite ChIPMunk papers in your publications (see the **Summary** section at the beginning of this guide).

ChIPMunk uses Apache Commons Math [http://commons.apache.org/math/] distributed under Apache License, version 2.0, and Trove [http://trove.starlight-systems.com] distributed under Lesser GNU General Public License.