

PACCMIT-CDS

Searching the coding region for microRNA targets [1]

Concise user's manual
(version 0.1)

Miroslav Šulc, Ray M. Marín, and Jiří Vaníček[†]

Laboratory of Theoretical Physical Chemistry
Institut des Sciences et Ingénierie Chimiques
Ecole Polytechnique Fédérale de Lausanne (EPFL)
CH-1015 Lausanne, Switzerland

[†]Electronic address: jiri.vanicek@epfl.ch

1	Concise user's manual	3
2	User's manual	5
2.1	Structure of the bundle	5
2.2	Installation	5
2.3	Software requirements	5
3	Basic usage	7
3.1	Input	7
3.1.1	Format of the input files	7
3.2	Output	8
3.3	Parameters of the algorithm	8
3.3.1	Shuffling 101	8
3.3.2	Shuffling constraints	9
3.3.3	Advanced options	10
3.4	Various options	11
4	Examples	12
4.1	Timing calculation	12
4.2	Printing out the gene sequence information	13
4.3	Displaying conservation information	14
4.4	Small working example	16
5	Supplementary utilities	17
5.1	Parallelization: SPLIT utility	17
5.1.1	Usage	17
5.1.2	Example	18

5.2	Histograms of the data: HISTO utility	18
5.2.1	Usage	19
5.2.2	Example	20
6	References	21

1 CONCISE USER'S MANUAL

Note that this chapter contains precisely the content of the *Concise User's Manual*.

PACCMIT-CDS is a program searching for microRNA targets within coding sequences. The program is written in C++ and is distributed in source form. The current version can be downloaded in a compressed form from <http://lcpt.epfl.ch>. Below we describe the basic usage of PACCMIT-CDS assuming a Linux-like operating system running on x86-64 architecture. As a courtesy to the authors, if you use this algorithm, please cite Ref. [1].

1. Download the tarball `paccmit-cds.tgz` from our website <http://lcpt.epfl.ch> and decompress it by means of the command

```
tar -xzf paccmit-cds.tgz
```

2. Compile the program with the command

```
make
```

3. The basic function of PACCMIT-CDS is taking two input files – one with coding sequences of the genes of interest and one with sequences of the miRNAs, and computing for each miRNA-gene pair **M-G** the probability (P_{SH}) that an interaction between **M** and **G** would occur by chance. The smaller this probability, the more likely is gene **G** the target for miRNA **M**.

There are two basic ways to run PACCMIT-CDS:

- if the conservation of target sites is not required:

```
./paccmit-cds \
-g ./data/genes_noncons_example.fa \
-m ./data/miRNAs_example.fa \
-i 8
```

- if the conservation of target sites is required (e.g., in at least 12 species):

```
./paccmit-cds \
-g ./data/genes_cons_example.fa \
-m ./data/miRNAs_example.fa \
-i 8 \
-x 27 \
-M">11"
```

The meaning of the program options employed above is:

- the option `-i 8` requests the P_{SH} values to be evaluated with precision 10^{-8} . Typically, the P_{SH} values are sufficiently resolved if the value of `-i` is set to

$$\log_{10}[\#\text{miRNAs} \times \#\text{genes}].$$

This value (or larger) should be used in productions runs. The run time of PACCMIT-CDS for large databases can be lowered by running the code in parallel (see [Section 5.2](#)).

- `-g` and `-m` options (required) specify the location of the gene and miRNA input files in FASTA format [see `genes_noncons_example.fa` and [Section 3.1](#) for more detailed description]
 - if the conservation is required, the switch `-x` determines the number of aligned sequences while the `-M` switch specifies the conservation mask [see [Subsection 3.3.2](#)]. The preceding example requires the conservation of the target sites in at least 12 species. For the precise format of the aligned sequences needed in this case, inspect the file `genes_cons_example.fa` and see [Section 3.1](#).
4. The output of PACCMIT-CDS is printed into files `stat_1ei.dat` for each i from 1 to the value set by the `-i` option. Each line of file `stat_1ei.dat` corresponds to a unique gene-miRNA pair and contains, in the sixth column, the P_{SH} value of this pair computed with precision 10^{-i} . In order to rank the predictions generated by either of the preceding two examples, simply sort the output file with:

```
sort -n -k6,7 stat_1e8.dat > ranked_predictions
```

The top predictions appear at the top of the file `ranked_predictions`, which is basically the final result of the PACCMIT-CDS algorithm. Note that generating the preliminary results in files with lower resolution of P_{SH} -values is useful since the calculation of final results can take a lot of time for large input files.

Finally, we note that files `human_mirna_v18.fa` and `human_hg18_28_species.fa` with all human miRNAs and with the aligned coding sequences of all human genes are available at <http://lcpt.epfl.ch> for more detailed tests.

Results published in Ref. [1] were generated by using these two files as input files in the preceding two examples.

References

- [1] R. M. Marín, M. Šulc, J. Vaníček, *RNA* accepted (2012).

2.1 Structure of the bundle

The program consists of several source `*.cc` and header `*.h` files which are stored in the “root” directory. The “production” tarball contains also several subdirectories:

- `include`
third party include files as detailed in [Section 2.3](#)
- `lib`
third party static libraries required to link the executables
- `data`
sample genome and microRNA data files (since the entire data files are quite large, we provide them in a separate archive `paccmit-data.tgz`)

2.2 Installation

We provide a minimalistic `Makefile` in order to facilitate the installation process. In principle the following command

```
make or make all
```

should compile all required source files and build final executables. Resulting executables are created in the directory in which the program was compiled. The main program can be launched by executing the command `./paccmit-cds`, options of which are detailed in [Chapter 3](#).

The attached tarball package has been tested with the GNU G++ (version 4.4.3) and Intel C++ (version 11.1) compilers under Linux (Debian) operating system running on x86-64 architecture.

2.3 Software requirements

The PACCMIT-CDS package relies partially on third party software, therefore it is necessary to set up the attached `Makefile` accordingly in order to be able to use PACCMIT-CDS properly. Following dependencies are enforced:

1. **zlib library** [2] – is utilized in order to enable reading of the (presumably large) files with genome information in compressed form as well as to write the output files generated by PACCMIT-CDS in the course of the computational process. Better performance is achieved with versions of the `zlib` library supporting internally the `gzbuffer` function. Static library `libz.a` (version 1.2.6 compiled for the `x86-64` architecture) is included in the PACCMIT-CDS bundle in the `./lib` subdirectory, while the necessary header files are stored in the `./include` subdirectory.
2. **RandomLib library** [3] – ensures the computational core of the PACCMIT-CDS package, namely the generation of the random numbers utilized in the implementation of the codon shuffling algorithm. The `RandomLib` library of Karney provides C++ interfaces for the Mersenne Twister random number generator MT19937 and as such should be far more reliable than the routines of the standard C/C++ library. For potential user's convenience, static version of the `RandomLib` library (compiled for the `x86-64` architecture) is included in the `./lib` subdirectory, while the necessary include files are stored in the `./include` subdirectory.

However, we strongly encourage potential reader/user to download the source codes of the above-mentioned libraries separately and compile them directly for his/her target architecture in order to gain optimal performance as well as prevent eventual linkage issues. In this case, the consequent modification of the `Makefile` can be easily done in two steps:

- set the `LIBRARY_PATH` variable to point to a directory containing required libraries. The switch `-Wl,static` of the GNU G++ compiler can be used in order to enforce utilization of static libraries.
- set the `INCLUDE_PATH` variable to point to a directory containing required header files.

The PACCMIT-CDS program accepts arguments from the command line in the standard UNIX-like fashion, *i.e.*, either in short or (if available) long form. In this section we include a concise list of the most important options. For description of the actual algorithm we refer the interested reader to the original publication [1]. Default values of parameters (if applicable) are shown in gray on the right.

3.1 Input

`-g, --glist=file`

specifies the path to the genome file. This file must contain at least one sequence per gene or several aligned sequences in case that binding site conservation should be taken into account. The latter case can be customized with the `-x` option (see below).

`-m, --miRNA=file`

specifies the path to the data file containing the miRNA sequences to be processed

3.1.1 Format of the input files

The input files are expected to be given in the standard FASTA format. The program expects the whole sequence for each gene to be on one line. When several aligned sequences are provided for one gene, each sequence must be on a different line (see example below). For the sake of completeness we show an excerpt of the sample input files provided with the PACCMIT-CDS package. Note that the PACCMIT-CDS program can read compressed (with the `gzip` utility) input files directly.

microRNA data

the data file containing the microRNA sequences consists of pairs of lines such as:

```
>hsa-let-7d-3p MIMAT0004484
CUAUACGACCUGCUGCCUUUCU
```


genome data

the genome file has a similar structure, nevertheless the “chunk” of data for each gene can consist of more lines if corresponding aligned sequences for different species are provided. A typical (truncated) example looks as follows

```
>ENST00000374123
AGCAAAGGGGGAAAGGGTCAGTCACCAGGGTGGGGCCAAGGCAGT
AGCAAAGGGGGAAAGGGTCAGGCCCCAGGGTGGGGCCAAGGCAGC
ATTAAAGGGGGAAAGGGTCAGTCACCATGGTGGGGGCCAGGCAGC
AGGGAAGGAAGAAGTGGCAGGCCCCAGGGGTAGGGTGAGGTAGT
-----CTCCTGGGGTGGGGTGGGGGGT
```

Here, the excerpt of the genome data file corresponding to the gene ENST00000374123 contains the human sequence (on the second row) together with other 4 aligned sequences.

3.2 Output

`--gzip`

write output files in compressed (gzip-ed) form. The resulting files are compatible with the gzip command present on most Linux systems.

3.3 Parameters of the algorithm

`-l, --mer-length=L`

default: L=7

this parameter controls the length of the seed sequence in the miRNA, *i.e.*, the number of consecutive nucleotides at the 5' end of the miRNA considered as the seed.

`-s, --start-offset=s`

default: s=2

starting position of the seed region counting from the 5' end of the miRNA. Default value of 2 thus corresponds thus to the seed starting at the second position in the miRNA.

3.3.1 Shuffling 101

`-i, --iteration-count=I`

perform I iterations (shuffle & analyze). At the I^{th} iteration, the P_{SH} values are evaluated with precision 10^{-I} .

`--randomize-genome`

shuffle (100× by default) the entire genome during the initialization phase, *i.e.*, before

actually starting to scan the randomized genome for seed matches

- r, --init-random-shuffles=R default: R=100
 override the default value of 100 shuffles for the initial randomization of the genome.
 If the --randomize-genome switch is omitted, this option is ignored.
- T, --shuffle-seed=T
 set the (integer) seed of the random number generator used in the shuffling procedure.
 The seed is generated randomly if this option is omitted.

3.3.2 Shuffling constraints

- x, --number-of-species=x
 specifies the number of additional aligned sequences requested for each gene. For instance if one is interested in considering conservation between human and mouse, the human and mouse sequences should be provided for each gene and x should be set to 1. If more than x sequences are provided in the genome file, the program will only read the first x + 1 sequences. Due to the implementation reasons, the program requires $x \leq 32$.
- M, --mask-rule="string"
 if x is greater than 0, string specifies the conservation rule, *i.e.*, given nucleotide is considered conserved if:

string	a given nucleotide is considered conserved if
<m	at most $m - 1$ species (including the reference species) match
>m	at least $m + 1$ species (including the reference species) match
=m	m species (including the reference species) match
: i_1, \dots, i_d	species i_1, \dots, i_d match ($d \leq s$ and $1 \leq i_j \leq s$)
@ i_1, \dots, i_d	(if and only if) species i_1, \dots, i_d match ($d \leq s$ and $1 \leq i_j \leq s$)

For example, the rule ": 1,2" should "pick out" exactly those sequence positions that contain the same nucleotides in the reference species sequence.

- initial-shuffling-preserves="method" default: method=cu+ps
 specification of the initial shuffling used to randomize the input genome data; method specifies the shuffling protocol

nothing	don't preserve anything
cu	preserve codon usage
ps	preserve protein sequence
cu+ps	preserve codon usage and protein sequence
ps+cu	
- shuffling-preserves="method" default: method=cu+ps
 specification of the shuffling as for the --initial-shuffling-preserves option. If only the option --shuffling-preserves is specified, the initial shuffling is performed using the same procedure.

3.3.3 Advanced options

This subsection details the procedure which we have employed in order to alleviate the overall computational costs. The idea consists in excluding some of the gene–miRNA pairs in the course of the computation, namely the pairs for which the P_{SH} value is already sufficiently converged and further refinement would be neither effective nor desirable.

The first method, which is simpler, is schematically shown in Figure 3.1. At n^{th} iteration (which evaluates the P_{SH} values with precision 10^{-n}), we divide the current *active* genome–miRNA pairs (those that are still being refined) into two groups according to the current P_{SH} value as alluded to in Figure 3.1. The group containing pairs with $P_{SH} \geq u \cdot 10^{-n}$ is considered sufficiently converged and is therefore “deactivated”: P_{SH} values of pairs in this group are not further refined.

in other words all the pairs belonging to this group are not subject to further refinement. It can happen that a particular gene will loose by this procedure all its miRNA “comrades” and hence can be excluded from the subsequent shuffling procedure with impunity. This leads to a significant speed-up of the calculation, since the shuffling comprises the most time-consuming part of the program.

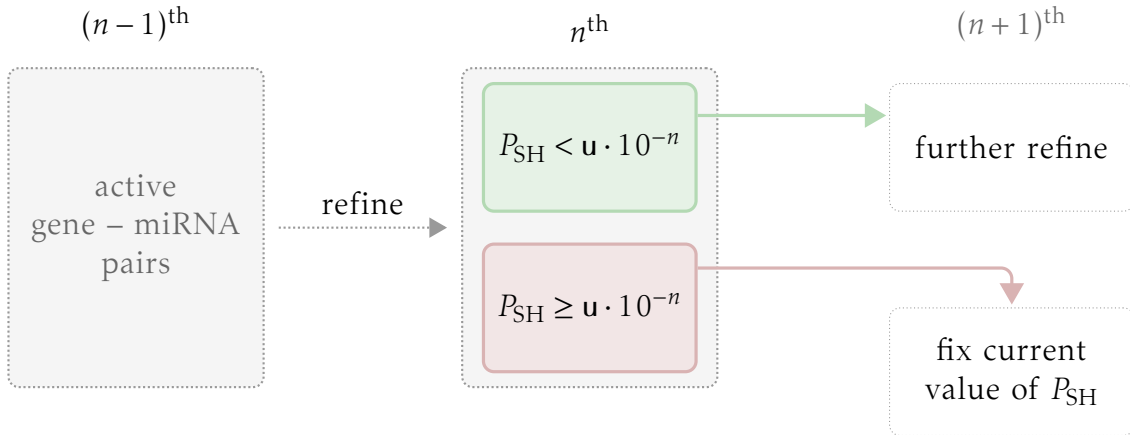


Figure 3.1: Strict (and simpler) method for excluding sufficiently converged gene–miRNA pairs in the course of the computation. At each “checkpoint”, *i.e.*, at n^{th} iteration, the gene–miRNA pairs are divided into two groups according to the current value of P_{SH} . The calculation continues only within the “green” group, *i.e.*, if the P_{SH} value is below a predefined threshold and hence needs to be evaluated more precisely.

A slightly generalized version of the algorithm introduced above is recorded in Figure 3.2. Here, the conceptual modification consists in dividing the set of all active gene–miRNA pairs into three groups as indicated. The “middle group” [not present in the algorithm of Figure 3.1] is refined only until the next checkpoint, *i.e.* until $(n+1)^{\text{th}}$ iteration. Because each gene is within this approach active for longer time, the total computational time will be in general longer than in the previous case. On the other hand, the generalized algorithm might prevent discarding some pairs prematurely.

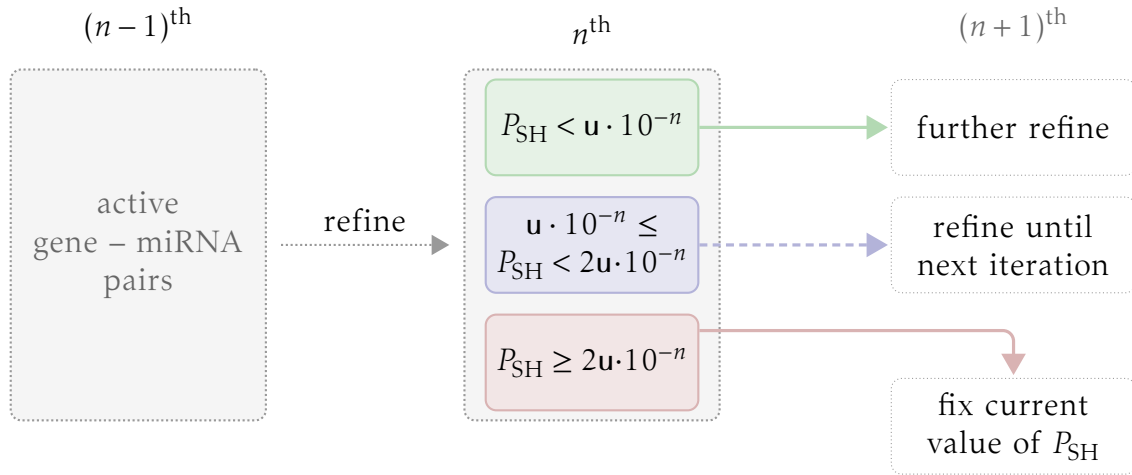


Figure 3.2: Modified method for excluding sufficiently converged gene–miRNA pairs in the course of the computation. At each “checkpoint”, *i.e.*, at n^{th} iteration, the set of gene–miRNA pairs is divided into three groups according to the current value of P_{SH} . The pairs having P_{SH} above a predefined threshold are excluded. The pairs belonging to the middle group are marked for exclusion at the next checkpoint, while the last group is simply further refined (without any marking).

The algorithm of Figure 3.2 is activated by default. This behavior can be altered by means of the following program options.

- `--no-filter`
do not use any “speed-up” techniques and refine all gene–miRNA pairs to the same (absolute) accuracy
- `--strict-filter`
use the algorithm of Figure 3.1 instead of the slower approach of Figure 3.2.
- `-u, --ulps=u` default: $u=4$
set the u parameter of the filtering procedure [see Figure 3.1 and 3.2]

3.4 Various options

- `-b, --benchmark`
for a given number of iterations, perform just the benchmark calculation, *i.e.*, shuffle & analyze the genome without producing any output, and measure the total elapsed time. This is useful for estimating the computational requirements.
- `--gzip`
write output files in compressed (gzip) form
- `-h, --help`
print an overview of the supported options

In order to illustrate currently implemented capabilities of the PACCMIT-CDS program package, this section provides a few examples showing how to use individual command line options discussed in [Chapter 3](#).

For this purpose we have used the data files accompanying the source code, namely the files:

- `./data/human_hg18_28_species.fa`
the genome file which contains also 27 additional aligned sequences
- `./data/human_mirna_v18.fa`
the miRNA data file encompassing 1919 sequences

These files can be downloaded separately [`paccmi t-data.tgz`] from <http://lcpt.epfl.ch>.

4.1 Timing calculation

Overall performance can be deduced from a simple timing calculation, which loads the entire genome file together with all provided miRNA sequences and performs given number of iterations, *i.e.*, rounds of shuffling and analysis. The corresponding command reads

```
./paccmi t-cds -b -i 3 \  
-g ./data/human_hg18_28_species.fa \  
-m ./data/human_mirna_v18.fa
```

where the `-b` options activates the timing mode. The output written by default to `stdout` might look as

```
22.11. 15:00:59.135 ( 0.000014 s): using random seed for the shuffling generator
+-----+
|                                     PACCMIT-CDS                                     v 0.1 alpha |
+-----+

./paccmi t-cds -b -g ./data/human_hg18_28_species.fa -m ./data/human_mirna_v18.fa -i3
random seed vector: [135384,1353592859,24956,2994753847,2012]
RNA random seed   : [1234]
+-----+

Parameters:
  shuffleSeed = -1
  rnaSeed     = 1234
  startOffset = 2
  mer length  = 7
  ulps        = 4
  iterations  = 10^3

Shuffling method:
  initial shuffling: (protein pres.: no, codon usage pres.: no)
  main shuffling:   (protein pres.: no, codon usage pres.: no)
```

time elapsed since last event

```

+-----+
22.11. 15:00:59.135 ( 0.000343 s): opening genome file './data/human_hg18_28_species.fa'
22.11. 15:01:03.308 ( 4.173268 s): estimated number of genes = 21426
22.11. 15:01:08.279 ( 4.970835 s): total count of genes: 21426
22.11. 15:01:08.279 ( 0.000018 s): average codon count per gene: 528.59
22.11. 15:01:08.279 ( 0.000041 s): opening miRNA file './data/human_mirna_v18.fa'
22.11. 15:01:08.280 ( 0.000271 s): estimated number of miRNAs = 1919
22.11. 15:01:08.281 ( 0.001494 s): total count of valid miRNA sequences: 1919
22.11. 15:01:08.281 ( 0.000009 s): count of valid unique miRNA sequences: 1552
22.11. 15:01:08.281 ( 0.000003 s): loaded miRNA sequences:
      id   code   name
      0   1C5D   hsa-let-7a-5p,hsa-let-7b-5p,hsa-let-7c,hsa-let-7d-5p,hsa-let-7e-5p,... (truncated)
      1   3ECC   hsa-let-7a-3p,hsa-let-7b-3p,hsa-let-7f-1-3p
      2   1EC4   hsa-let-7a-2-3p,hsa-let-7g-3p
      3   36CC   hsa-let-7d-3p
      4   16CC   hsa-let-7e-3p

```

number of genes found in the input file

some of the miRNA sequences are generally non-unique

current time

miRNA sequences corresponding to the same L-mer code

Once the calculation is finished, the program informs the user and terminates.

```

22.11. 15:01:08.747 ( 0.367589 s): starting benchmark 10^03 iterations
22.11. 15:04:29.722 ( 3.349580 m): benchmark finished
22.11. 15:04:29.722 ( 0.000019 s): total benchmark time: 200.974803 s
22.11. 15:04:29.722 ( 0.000004 s): finished

```

The calculation took (using 1 core of a main stream desktop computer) about 3 minutes. This justifies the need for algorithms along the lines of the approaches discussed in [Subsection 3.3.3](#), since 10^4 iterations within the same setting would take half an hour, while 10^5 iterations would necessitate already more than 2 days.

The excerpt of the log included above contains other useful pieces of information (highlighted above). Most importantly, note that a given seed region in the miRNA sequences (specified by the `-s` and `-l` switches discussed on [p. 8](#)) can lead to duplicities among the miRNA sequences present in the input file. The PACCMIT-CDS program then groups the “seed-equivalent” miRNA sequences and prints out corresponding collective label. From the log excerpt we see that in this case the number of miRNA sequences was reduced from 1919 to 1552.

4.2 Printing out the gene sequence information

In order to verify the loaded gene sequences, PACCMIT-CDS provides a `-p` switch which asks the program to print the amino acid sequences.

Example:

```
./paccmit-cds -g ./data/ENST00000374123.dat -m ./data/human_mirna_v18.fa \
-i3 -p ./genes
```

Directory `./genes` is assumed to be accessible for writing. Since the genome file in its entirety is certainly too big for this particular demonstration, we have extracted one particular

gene, ENST00000374123, and saved its sequence (together with other aligned sequences) into a separate file.

Before performing 3 iterations for this gene, the command above saves the amino acid information of this gene into the file `./genes/ENST00000374123.dat` (its content is shown below). Should more genes be present in the input, a separate file is created for each gene.

```
Gene 'ENST00000374123':
  AA usage:
    arginine: 0
    leucine: 0
    serine: 1
      7
    alanine: 0
    glycine: 6
      2 3 5 9 11 13
    proline: 1
      8
    threonine: 0
    valine: 0
    isoleucine: 0
    asparagine: 0
    aspartic acid: 0
    cysteine: 0
    glutamic acid: 0
    glutamine: 2
      6 12
    histidine: 0
    lysine: 2
      1 4
    phenylalanine: 0
    tyrosine: 0
    methionine: 0
    tryptophan: 1
      10
sequence:
  0 1 2 3 4 5 6 7 8 9
000: AGC AAA GGG GGA AAG GGT CAG TCA CCA GGG
010: TGG GGC CAA GGC AGT
```

The structure of the file is rather self-explanatory. It lists the number of occurrences of each AA in the given sequence. The corresponding positions in the sequence are also provided.

4.3 Displaying conservation information

It can be useful to visualize the conservation mask which affects matching of miRNA seeds against individual genes and enters the code *via* the `-M` and `-x` switches discussed in [Subsection 3.3.2](#).

```
./paccmit-cds -g ./data/ENST00000374123.dat -m ./data/human_mirna_v18.fa \
-x27 -M">11" -i3 -P./mask.dat
```

>ENST00000374123
AGCAAAGGGGGAAAGGGTCAAGTCACCAGGGTGGGGCCAAGGCAGT
AGCAAAGGGGGAAAGGGTCAAGTCACCAGGGTGGGGCCAAGGCAGC
ATTAAAGGGGGAAAGGGTCAAGTCACCAGGGTGGGGCCAAGGCAGC
AGGGAAAGGAAGAAGTGGCAGGCCCCAGGGGTAGGGTGAGGTAGT
-----CTCCTGGGGTGGGGTGGGGGGGT
AGT-----GGTCAG-----
AGT-----GGTCAT-CACAGGGCTGAGGGTGAGATGGT
GG-----

AGGAAAGGGGAAAAGTGTCAATCACCTGGGGTGGGCTGAGGCCAT
AGGAAAGGGGAAAGTGGTCAAGTCACCAGGGCGGGCTGAGGTAGC
GGGAAAGGAGGAAGTGGGCAGTCACCTGGGGCGGGCCGAGTAGC
AGGAAAGGCAGAAGTGGTCTGACAACATGGACGGAAGT
AGGAGAGGGGAA-GTGGTCAGCACCTGGGG-----
AGGAAAGGGGGACATGGCGGT-----
GGGAATGTGCCCTGTGGCTGGT-----

The generated file `./mask.dat` contains two lines for each gene, namely the gene designation (label) and a binary representation of the conservation mask, *i.e.*, 0's represent those nucleotide positions, where the mask criterion is not fulfilled and *vice versa*.

which is consistent with the graphical representation included above.

4.4 Small working example

```
./paccmit-cds -g ./data/human_hg18_28_species.fa -m ./data/human_mirna_v18.fa -i4
```

loads the entire genome file (included in the PACCMIT-CDS package) as well as all miRNA sequences and request to perform 4 iterations employing the filtering procedure of [Subsection 3.3.3](#). Moreover, the option `--gzip` enforces compression of the output files.

After the j^{th} iteration, PACCMIT-CDS creates an output file called `stat_1ej.dat`. The example below shows the excerpt of `stat_1e4.dat`.

genes					P_{SH}	miRNA sequences
ENST00000369381	A	1	41	3!	4.100000e-02	hsa-miR-4735-5p
ENST00000369381	A	1	39	3!	3.900000e-02	hsa-miR-4742-3p
ENST00000369381	A	4	0	4+	0.000000e+00	hsa-miR-4753-3p
ENST00000369381	A	1	56	3!	5.600000e-02	hsa-miR-4762-3p
ENST00000369381	A	1	60	3!	6.000000e-02	hsa-miR-4768-5p
ENST00000369381	A	2	9	4~	9.000000e-04	hsa-miR-4775
ENST00000369381	A	1	49	3!	4.900000e-02	hsa-miR-4778-3p
ENST00000369381	A	2	15	4!	1.500000e-03	hsa-miR-4436b-5p
ENST00000369381	A	1	56	3!	5.600000e-02	hsa-miR-4781-3p
ENST00000369381	A	1	58	3!	5.800000e-02	hsa-miR-4999-5p
ENST00000369381	A	1	63	3!	6.300000e-02	hsa-miR-5192
ENST00000369381	A	1	58	3!	5.800000e-02	hsa-miR-5193
ENST00000369381	A	1	55	3!	5.500000e-02	hsa-miR-5582-3p
ENST00000369381	A	1	38	3!	3.800000e-02	hsa-miR-5692a
ENST00000369381	A	1	40	3!	4.000000e-02	hsa-miR-5694
ENST00000369382	E	1	142	4!	1.420000e-02	hsa-miR-200b-3p, hsa-miR-200c-3p, hsa-miR-429
ENST00000369382	E	1	12	3!	1.200000e-02	hsa-miR-324-5p
ENST00000369382	E	1	98	4!	9.800000e-03	hsa-miR-491-3p

The gene and miRNA sequence designations (labels) are written out to the first and seventh columns, respectively, while the current value of P_{SH} is stored in the sixth column.

The columns number 2 and 5 are pertinent to the filtering procedure of [Subsection 3.3.3](#). Specifically, the second column contains either character A or E depending on the fact whether the corresponding gene has been already excluded (E) or is still active (A). By terming a particular gene as “being excluded” (from the calculation) we mean that there are no pairing miRNA sequences which (in combination with the gene under consideration) would require further refinement of the P_{SH} value.

The “fate” of each gene in the course of the calculation is recorded in the fifth column. The meaning of the (perhaps a bit cryptic) values is following. If the flag has a value as indicated in the first column of the table below, then the P_{SH} value of a particular gene–miRNA pair:

- n! was finalized (excluded from further refinement) at n^{th} iteration
- n+ is at $(n+1)^{\text{th}}$ iteration still subject to further refinement
- n~ will be finalized at $(n+1)^{\text{th}}$ iteration (in case of the modified algorithm of [Subsection 3.3.3](#))

Finally, note that the current implementation skips this filter at 1st and 2nd iterations.

5 SUPPLEMENTARY UTILITIES

5.1 Parallelization: SPLIT utility

The purpose of this utility is to facilitate calculations in a “parallel” fashion, where the parallelism consists in splitting the input genome file into several parts which can be then processed independently.

The implementation of PACCMIT-CDS, as described above, scales linearly with the total number of nucleotides contained in the genome file (and also with the number of corresponding aligned sequences).

A drawback of a parallelization based on a naïve division of the genome file into several groups would be that so produced groups might differ significantly in the total numbers of nucleotides contained in different groups. This would in turn spoil the efficiency of the entire “parallel” approach.

That is where the SPLIT utility sets in. To be more specific, SPLIT tries to divide the genome file into several groups with similar nucleotide counts. We have implemented a very simple approach [described below] which is definitely far from optimal, nevertheless it should be still superior to the naïve method.

Our implementation simply loads the genome file, sorts the genes according to the length of the nucleotide sequence and employs the “card dealer” method, *i.e.*, the longest gene is attributed to the first group, the second longest gene to the second group and so forth. Formally, if one labels the sorted genes by index $1 \leq i \leq M$, then the i^{th} gene goes into the j^{th} group, where $j = (i - 1) \% N + 1$ with N denoting the total number of groups, *i.e.*, $1 \leq j \leq N$. Moreover, current implementation performs two passes through the genome file in order to obviate the need to hold the (possibly quite huge) genome file in memory.

5.1.1 Usage

The utility is invoked from the command line as follows:

```
./split [OPTIONS]
```

-g, --genome-file=path

path to the (eventually gzip-ed) genome file in FASTA format

-n, --number-of-groups=N

number of groups (should be between 2 and 256)

-c, --number-of-cores=c

default: c=1

use c cores (threads, to be more precise) for the compression of the individual group

files. The program does not allow to use more threads than $C - 1$, where C denotes the number of cores of the machine executing the utility (the value of C is determined by the utility automatically in run-time)

- p, --prefix=pattern
output file names will be generated as `pattern_%03d.gz`¹
- h, --help
print short usage instructions

5.1.2 Example

The command

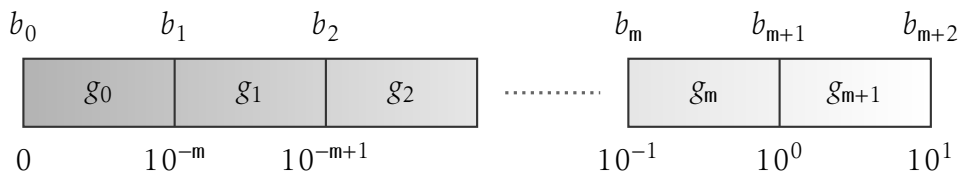
```
./split -g"genome.list" -n16 -c3 --pattern="part_"
```

will look for the `genome.list` file in the current directory, split it into 16 (almost) equally sized groups, and write the groups of genes to files `part_000.gz`, `part_001.gz`, ..., `part_016.gz`. Saving and compression of the resulting files will be done in 3 threads provided that the machine executing this command has at least 4 cores.

5.2 Histograms of the data: HISTO utility

The second small utility is an implementation of a naive algorithm producing histogram out of the output file(s) containing the P_{SH} values for the gene-miRNA pairs under considerations. The output file is assumed to be in the format detailed in the previous section, *i.e.*, the P_{SH} values are stored in sixth column. The algorithm proceeds in the following steps:

1. for a given value of m , construct histogram bins as



2. load the PACCMIT-CDS output file and for each row, *i.e.*, value of P_{SH} determine index j such that

$$b_j \leq P_{SH} < b_{j+1}.$$

Since $0 \leq P_{SH} \leq 1$ by definition, then $0 \leq j \leq m + 1$. Consequently, increase the counter of the j^{th} group, *i.e.*, $g_j \leftarrow g_j + 1$.

¹*i.e.*, the first two group files will be named `pattern_000.gz`, `pattern_001.gz` and so forth

3. output the values $\{g_j\}_{j=0}^{m+1}$.

Although an implementation in some **UNIX**-like scripting utility would be straightforward, the resulting script would be typically quite slow for large output files. There are two advantages provided by the **HISTO** utility. First, **HISTO** is implemented in **C++**. Secondly, **HISTO** is able to process several output files at once and produce an overall histogram. Such files can be obtained, for example, by splitting the input into several parts each of which is then processed independently, as discussed in [Section 5.1](#) on parallelization.

5.2.1 Usage

The utility is invoked from the command line as follows:

```
./histo [OPTIONS]
```

- `-f, --stat-file="path"`
path to the (eventually gzip-ed) output file
- `-d, --stat-dir="target"`
instead of specifying a single input file, scan the directory `target` containing the output files
- `-o, --output-file="target"`
if specified, the file `target` will contain the (merged) overall statistics from all processed output files
- `-p, --prefix="string"`
search only for files the name of which starts with prefix `string`. This option is effective only in directory mode enabled by the `-d` switch.
- `-s, --suffix="string"`
search only for files the name of which ends with suffix `string`. This option is effective only in directory mode enabled by the `-d` switch. In case both options `-p` and `-s` are specified, the `-p` option takes precedence.
- `-m, --min=m` default: `m=9`
sets the `m` parameter as elaborated above (*i.e.*, the value of `m` determines the requested number of bins)
- `-h, --help`
print short usage instructions

5.2.2 Example

The command

```
./histo -f"stat_1e3.dat.gz" -h"histo.dat" -m4
```

loads the compressed output file `stat_1e3.dat.gz` (located in the current directory) and creates an histogram consisting of bins with boundaries at $0, 10^{-4}, 10^{-3}, \dots, 1, 10$.

Typical output might then look like

```
5.00000e-05 91638
5.50000e-04 0
5.50000e-03 235424
5.50000e-02 1654334
5.50000e-01 1691010
#0
```

In words, the program prints out the midpoints of individual bins with corresponding populations. The population of the last bin, *i.e.*, pairs with $P_{SH} = 1$, is printed out on the last line after the `#` sign. However, since these pairs are not written into the output, the value is correctly 0.

As compared to a straightforward `gawk` implementation

```
BEGIN{
    minPower = 9; breaksCnt = minPower + 3

    histo[0] = 0; breaks[0] = 0.0; breaks[breaksCnt - 1] = 10.0

    for(i = breaksCnt - 2; i > 0; i--){
        histo[i] = 0; breaks[i] = breaks[i + 1] / 10.
    }
}

{
    val = $6*1.0; i = 1;
    while(val >= breaks[i] && i < breaksCnt){ i++ }; i = i - 1
    histo[i]++;
}

END{
    for(i = 0; i < breaksCnt - 2; i++){
        print (breaks[i] + breaks[i+1])/2, histo[i]
    }
    print "#", histo[breaksCnt - 2]
}
```

the `HISTO` utility is typically faster. For the input files provided with the PACCMIT-CDS package, the overall output file consists of, roughly, $3.5 \cdot 10^6$ lines. While the execution time of the `gawk` script on a main stream desktop computer takes about 30s, the `HISTO` utility provides an equivalent result in less than 1.5s including the internal decompression.

6 REFERENCES

- [1] R. M. Marín, M. Šulc, J. Vaníček, *RNA* accepted (2012).
- [2] J.-l. Gailly, M. Adler, *ZLIB library* (<http://www.zlib.net>).
- [3] C. Karney, *RandomLib library* (<http://sourceforge.net/projects/randomlib/>).