

RNAProfile 2.0

User Manual

Giulio Pavesi

April 3, 2004

Compiling the Program

RNAProfile comes as a .tar.gz package. It is written in C++, and has been successfully compiled with `gcc` under different Unix/Linux/Cygwin platforms. To extract the files:

```
$ gunzip rnaprofile.tar.gz
$ tar xvf rnaprofile.tar
```

A directory named "RNAProfile2.0" should appear. It should contain a directory named "src" and another one named "examples". To compile the program:

```
$ cd RNAProfile2.0
$ cd src
$ make
```

An executable file named "rnaprofile" has been placed in the parent directory (RNAProfile1.1). In case `gcc` is not available on your computer, edit the Makefile file and replace it with your C++ compiler.

```
$ cd ..
```

to go back one level. The executable should be there.

Running the Program

To run the program:

```
$ ./rnaprofile -f <filename>
```

if you are looking for hairpin motifs, that's basically it. The input file should contain the input sequences, in FASTA format if the sequences have not been pre-processed, or in one of the supported preprocessed formats (see later on). There's a number of additional flags/parameters, with supposedly optimal default settings determined in our experiments, that can anyway be modified. The default settings have been defined mostly to speed up the execution of the program, with good results.

-A <int>: run the program in "all versus all" mode. This is explained in the paper in the "finding motifs shared by a few sequences" section. At each iteration, all regions are compared against all the other regions from different sequences (instead of processing just one sequence at each iteration). The algorithm stops when profiles containing <int> regions have been generated. The best profiles of each iteration with up to <int> regions are anyway output. All the other parameters (except the ones that work on the random sequence order that no longer make sense) have the same effect also in this case.

-H <int>: this is used to define the secondary structure template. The template is defined according to the number of hairpins the secondary structure associated with the candidate regions should contain. Default is 1 (the default is a single hairpin). Look also at **-l**, **-L**, and **-e** parameters.

-v : verbose mode. It outputs additional information while running, like the number of candidate regions for each input sequence.

-r yes : turn on the random picking order of the sequences (default: process the sequences in the order they appear in in the input file)

-r <seed> : use <seed> (integer number) as seed for the random picking order of the sequences. Runs with the same seed have identical output.

-P <int> : number of profiles kept at each step/output by the program. Default is 100.

-l <int>: minimum length of the regions to be considered. Default is 20 for single hairpin motifs, 30 for two hairpin motifs, 40 for three, and so on. These numbers work well with single or two hairpins. To save time, it should be increased in case of more complex structures (involving larger regions). It should be reduced when looking for small hairpins (whose region is shorter than 30 nts).

-L <int> : maximum length of the regions to be considered. Default is 40 for one hairpin, 60 for two, and so on.

-o <filename> : just process the sequences and save the candidate regions in a file. When this option is used, the program stops after the first phase (generation of candidates according to the parameters set).

-s <int>: do not compare regions or profiles whose size difference is larger than <int>. Used to speed up the program, avoiding alignments very unlikely to produce good results. Default is 10.

-i <int>: run the program consecutively for <int> times. A different random seed should be used for each run, so to get different results at each run (use the '-r yes' parameter in combination with this one!).

-e <float>: consider only candidate regions fitting the structural template whose structure has an energy lower than <float>. Default is -1 for single hairpins, -10 for two hairpins, and so on. Makes sense if the <float> specified is lower than zero.

-p <int>: at each step, no more than <int> of the best profiles saved can be generated by the same starting profile. Default is 10. Used to avoid premature convergence.

-params <filename>: read the alignment parameters from a file, instead of using the ones described in the paper. The file should have this format:

Mll	Mlu	Muu	mll	mlu	muu	Gl	Gu
2	-15	.5	-1	-15	-.5	-2	-2

Two lines, and elements on each line are separated by a TAB character. Mll is the score for the match between two unpaired nucleotides (same letter lowercase-lowercase). Mlu match between paired and unpaired (same letter in lowercase-uppercase). mll is the mismatch between two lowercase. muu is the mismatch between two uppercase. mlu is the mismatch between an uppercase and a lowercase. Gl is the gap penalty for aligning a gap with a lowercase letter. Gu is the gap penalty for aligning a gap with an uppercase letter. A sample parameters file is provided in the examples directory (just edit it if you want to change parameters).

Working on Pre-processed Sequences

There's also the possibility of running the program on a pre-processed dataset, instead of using the standard region selection method provided with the algorithm. In this case, the input file should contain:

- The input sequences, in FASTA format
- The candidate regions with their secondary structure in bracket notation

The sequences are separated from their respective regions. The regions of sequence **seqname** should come after a line with exactly the same FASTA header of **seqname** where the > symbol in the header is replaced by ! (exclamation mark). For example:

```

> Sequence 1
CAGTCAGTACGTCTGACAGTCAGTACATGCTCGATGGTACGTATGCATGCGTGT
CATCAGTCTGAGTCAGTACTGACGTAGTCAGTCTGACTGACGTATGCAGTCTGA
! Sequence 1
GTTTCGTCCTCAGTGCAGGGCAAC
(((.((((.....))))))
AACTTCAGCTACAGTGTAGCTAAGTT
(((((((((.....))))))))))
CCACAGGCTCAGTGTGGTCTTGG
(((.((((.....))))))
GCCTTCTGCACCAAGTGTGTAAAGGC
(((((((((.....))))))))))
GCCTTCTGCGCCAGTGTGTAAAGGC
(((((((((.....))))))))))
TAATTGCAAACGCAGTGCCGTTTCAATTG
(((((((((.....))))))))))
> Sequence 2
CTACTGACAGTCAGTCATGCGTACAGTGTGAGTCATGCAGTCAGTACCGTACGTA
CATGACGTCATGCATGCATGCAGTCAGTCATGCAGTCATGCATGCATGCAGTCAG
! Sequence 2
CCACAGGCTCAGTGTGGTCTTGG
(((.((((.....))))))
GCCTTCTGCACCAAGTGTGTAAAGGC
(((((((((.....))))))))))
GCCTTCTGCGCCAGTGTGTAAAGGC
(((((((((.....))))))))))
TAATTGCAAACGCAGTGCCGTTTCAATTG
(((((((((.....))))))))))

```

and so on. IMPORTANT: the sequences are used only to compute the background frequency of nucleotides for the scoring of profiles. The regions of a sequence are not checked against the sequence itself. Thus, you can put basically anything instead of the sequences. If you just put ACGT, the program will assume that the regions come from a sequence with uniform nucleotide composition, even if a single region is longer than the sequence itself (perhaps this is useful if the regions are taken, say, from a whole genome. You do not need to put the whole genome in the input file). The order in which the sequences and regions appear in the file is not important. For example, you can put all the sequences at the beginning, and the regions after them. Or, the regions before the sequences. It does not matter. If an orphan sequence is found (thus, without the list of regions), the algorithm assumes that it has to be processed with its method (so you can use a combination of processed/unprocessed sequences in the input). Instead, if a list of regions is found without a reference sequence, an error message is displayed.

There are some additional parameters you can use in this case:

-e <float>: as in the previous case, the algorithm computes the free energy of the structures associated with the regions, and discards those whose energy is higher than float. **IMPORTANT:** only the energy of the structure is checked, NOT its optimality.

-O: with this flag, **ONLY** the regions whose structure is optimal are kept. That is, the program folds each of the regions by itself, and compares the structure found in the file with the one it obtained. Only those regions whose two structures match are kept and passed to the program. The others are discarded.

Alternatively, the program can start from one or more existing profiles. The input format for a profile is:

```
#5.7844 7
a      0.0    0.0    0.0    0.0    0.0    0.0    ...
c      0.0    0.0    0.0    0.0    0.0    0.0    ...
g      0.0    0.0    0.0    0.0    0.0    0.0    ...
t      0.0    0.0    0.0    0.0    0.0    0.0    ...
A      0.0    0.0    0.9    1.0    0.0    0.0    ...
C      0.0    0.0    0.0    0.0    0.1    0.0    ...
G      1.0    0.9    0.1    0.0    0.9    1.0    ...
T      0.0    0.1    0.0    0.0    0.0    0.0    ...
-      0.0    0.0    0.0    0.0    0.0    0.0    ...
```

The numbers after the # symbol are the score of the profile (see the paper) and the number of regions that were used to build it. This is exactly how profiles are output by the program, so you can recycle them by cutting and pasting to process new data. An input file can thus contain one or more profiles, and sequences, either pre-processed or not. The algorithm, instead of starting from the pairwise alignment of two sequences, will start from the profiles found in the file. All the parameters work as usual.

The Output

When running, the program outputs on the screen some stuff concerning the run, like which sequence is being processed, how many sequences it has to process before the end, the best score found so far, and so on. The real results are saved into a file, whose name is output on the screen at the end of the run. The output file looks like:

```
ALIGNMENT RESULTS
Input file: mouse+human.ferritin.fna
Number of profiles saved at each step: 100
Max number of profiles originating from the same profile (or region): 10
Region minimum length: 20
Region maximum length: 40
```

```

Energy threshold: 0
Max difference in length between regions: 10
Random alignment: yes (Seed used: 1065538283)

Best profiles:
Profile 1. Score: 2.66
(profile data here)

>gi|33859501|ref|NM_009653.1| Mus musculus aminolevulinic acid synthase 2, mRNA
gGTTcGTCCtcaagtgcAGGGCAACa
.((((((((.....))))))). (E: -7.2 Fitness: 4.6)
>gi|6753913|ref|NM_010240.1| Mus musculus ferritin light chain 1 (Ftl1), mRNA
cTTGcTTCAAcagtgtTTGAACGCa
.((((((((.....))))))). (E: -1.8 Fitness: 9.0)
>gi|6753911|ref|NM_010239.1| Mus musculus ferritin heavy chain (Fth), mRNA
cCTGcTTCAAcagtgcTTGAACGCa
.((((((((.....))))))). (E: -4.4 Fitness: 8.1)
>gi|20149497|ref|NM_000146.2| Homo sapiens ferritin, light polypeptide (FTL), mRNA
cTTGcTTCAAcagtgtTTGGACGCa
.((((((((.....))))))). (E: -1.3 Fitness: 9.8)
>gi|4557298|ref|NM_000032.1| Homo sapiens aminolevulinate, delta-, synthase 2
cGTTcGTCCtcaagtgcAGGGCAACa
.((((((((.....))))))). (E: -7.5 Fitness: 6.3)
>gi|507251|gb|L20941.1|HUMFERRITH Human ferritin heavy chain mRNA, complete cds
cCTGcTTCAAcagtgcTTGGACGCa
.((((((((.....))))))). (E: -3.9 Fitness: 8.8)
>gi|806340:c862-1 H.sapiens (24) Ferritin H pseudogene
aATTCTTTatttGAAGGAATg
.((((((((.....))))))). (E: -4.5 Fitness: -3.6)
>gi|182512|gb|J04755.1|HUMFERHX Human ferritin H processed pseudogene, complete cds
ctTAGTCATTgccatGATGACTGca
..((((((((.....))))))). (E: -7.5 Fitness: -39.7)
>gi|806342|emb|X80336.1|HS5FERHPE H.sapiens (5) Ferritin H pseudogene
TTCTTCaCCaaTCtcatGAGGaGAGGGA
((((((((.....))))))). (E: -5.8 Fitness: -321.6)

```

At the beginning, there are the input parameters. Then, the list of the highest scoring profiles, with the score, and the list of regions that have been used to build each profile, as well as the corresponding folding energy (E). Below each region, you can also see where the algorithm put gaps when building the alignment. **9 Sequences used** means that the algorithm has found 9 sequences (preprocessed or not) in the input file. **Unknown: 0** means that the algorithm started from scratch instead of from existing profiles (built using a number of unknown sequences) in the input file. Each instance comes after the name of

the sequence it was taken from. It shows the region and its secondary structure. Next to each instance, the (E:) value gives the energy associated with the structure, followed by the fitness value of the instance (see the paper for details on how the fitness is computed). The rationale is: positive or zero point something (also minus zero point something) fitness value means that the region is surely an instance of the motif described by the profile. Negative values are instead suspicious. The more negative, the more unlikely to be a real instance of the motif.

Contact

For comments, praise, bug reports (especially, please!), donations, contact me (Giulio Pavesi) at pavesi@disco.unimib.it.