`rlsim` – a package for simulating RNA-seq library preparation with parameter estimation

Botond Sipos

`sbotond@ebi.ac.uk`
`http://sbotond.github.com`

August 15, 2013

 [1]

Contents

1 Introduction

1.1 What is the `rlsim` package?

The `rlsim` package is a collection of tools for simulating RNA-seq library construction, aiming to reproduce the most important factors which are known to introduce significant biases in the currently used protocols: hexamer priming [22], PCR amplification [20, 19] and size selection [25]. It allows for a systematic exploration of the effects of the individual biasing factors and their interactions on downstream applications by simulating data under a variety of parameter sets.

The implicit simulation model implemented in the main tool (`rlsim`) is inspired by the actual library preparation protocols and it is more general than the models used by the bias correction methods (e.g. [26], [23]), hence it allows for a fair assessment of their performance.

Although the simulation model was kept as simple as possible in order to aid usability, it still has too many parameters to be inferred from data produced by standard RNA-seq experiments. However, simulating datasets with properties similar to specific datasets is often useful. To address this, the package provides a tool (`effest`) implementing simple approaches for estimating the parameters which can be recovered from standard RNA-seq data (GC-dependent amplification efficiencies, fragment size distribution, relative expression levels).

The key features offered by the package are the following:

- Simulation of priming biases loosely based on a nearest-neighbor thermodynamic model.
- Exact simulation of PCR amplification on the level of individual fragments (consistent across expression levels, no approximations).
- Fragment-specific amplification efficiencies determined by GC-content and length.

- Possibility to simulate PCR and sampling pseudo-replicates.
- Simulation of size selection and polyadenylation with flexible target distributions.
- Estimation of GC-dependent amplification efficiencies from real data, relying on assumptions about locality of biases and the mean efficiency of the fragment pool.
- Estimation of relative expression levels.
- Estimation of empirical fragment size distribution, model selection between normal vs. skew normal distributions.
- Able to simulate experiments on the human transcriptome over a wide range of expression levels on a desktop machine.

The `rlsim` package does not simulate the exact physical and chemical processes which take place during library construction, as such a simulation model would involve too many hidden parameters with not enough information in the data about their possible values.

1.2 Citing the package

An associated manuscript is in preparation, meanwhile the package should be cited as:

> Botond Sipos, Greg Slodkowicz, Tim Massingham, Nick Goldman (2013) *Realistic simulations reveal extensive sample-specificity of RNA-seq biases* **arXiv**:1308.3172

1.3 Getting more help

The `BioStar` Q&A forum (`http://www.biostars.org`) is an excellent place to get additional help. The author of the package will monitor the posts having the `rlsim` tag.

1.4 Dependencies and installation

1.4.1 Dependencies

The package runs on 64-bit GNU/Linux operating systems. The `rlsim` tool is written in `Go` [3] and shipped as a statically linked executable for the `amd64` Linux platforms, hence it has no dependencies.

WARNING: The `rlsim` tool can be built for other architectures supported by the `Go` compiler, however only the `amd64` architecture is supported and the 32-bit binaries might not work properly.

The parameter estimation tool (`effest`) and the additional tools are written in `Python 2.x` and depend on a couple of packages:

- `numpy >= 1.6.2` [7]
- `matplotlib >= 1.1.0` [6]
- `scipy >= 0.10.1` [12]
- `biopython >= 1.60` [1]
- A modified version of the `HTSeq` [4] package available from the GitHub repository under `https://github.com/sbotond/rlsim/tree/master/misc`.

  WARNING: The official releases of the HTSeq package contain a bug causing segmenation fault when parsing certain paired-end datasets. Please use the modified version from the `rlsim` repository!

These packages (with the exception of the modified HTSeq) are readily installable from the `Python` Package Index [8] using the `pip` tool by issuing the following command:

```
pip install numpy matplotlib scipy biopython
```

The additional tool `plot_cov` tools depend on `samtools` [11]. Simulating Illumina sequencing of the fragments can be done by using the `simNGS` package [13] (recommended) or any other sequencing simulator.

1.4.2 Installing the latest release

The release tarballs can be obtained from the `rlsim` download [9] page on `GitHub`:

```
wget https://github.com/sbotond/rlsim/blob/master/releases/rlsim-latest_amd64.tar.gz?raw=true t.tgz
```

The unpacked release directory contains the following files:

- `bin/` – directory containing the executables:

  `rlsim` – the main tool simulating library construction

  `effest` – tool for estimating selected parameters from real datasets

  `sel` – tool for sampling expression levels

  `plot_rlsim_report` – tool for plotting the `rlsim` report

  `pb_plot` – tool for visualising sequence biases

  `cov_cmp` – tool for comparing coverage trends across datasets

  `plot_cov` – tool for plotting transcript read coverage colored by reference base

- `COPYING` – GPL v3 licence
- `README.md` – short instructions in markdown format
- `rlsim_manual.pdf` – package manual

The executables under `bin/` can be installed by copying them to a directory listed in the `$PATH` environmental variable.

1.4.3 Building from source

Build dependencies:

- The package is built using `make` in a standard `Linux` environment.
- Building the `rlsim` tool requires the `Go` compiler which can be installed as described on the projects website [5].
- A standard LaTeX installation with `pdflatex` has to be present in order to produce the package documentation.
- Regenerating the test datasets needs the `bwa` [2] and `samtools` [11] commands to be installed.

The package source can be obtained by cloning the `GitHub` repository [10] and built by issuing `make` in the top level directory:

```
git clone https://github.com/sbotond/rlsim.git
cd rlsim
make
```

A tarball can be built by issuing:

```
make release
```

1.5 Examples

The following basic examples can be run from the `src/` directory in the package source tree:

- Re-sample expression levels from a mixture of gamma distributions with two components with mean 5,000 and 10,000:

  ```
  $ ../tools/sel -d "0.5:g:(5000, 0.1) + 0.5:g:(10000, 100)" \
  test/basic/test_transcripts.fas > my_transcripts.fas
  ```

  The output `my_transcripts.fas` is a Fasta file annotated with the expression levels:

  ```
  >ENST00000371588$9430
  GCTTCCGGCATCTGGCTCAGTTCCGCCATGGCCTCCTTGGAAGTCAGTCGTAGTCCTCGCAGGTCTCGGCGGGAGCTG ...
  ...
  ```

- Simulate 10,000 fragments with default parameters, plot `rlsim` report:

  ```
  $ ./rlsim -n 10000 my_transcripts.fas > frags.fas
  $ ../tools/plot_rlsim_report
  ```

  The output file `frags.fas` contains the simulated fragments:

  ```
  >Frag_0 ENST00000374005 (Strand - Offset 1883 -- 2298)
  AGAGAATAGAGGGTAGAAGGGAAATTCTTGGCACCTGGACTAGAGTGAGATAAAAGGAGAGTAGGAAAGCAGTGA ...
  ...
  ```

- Simulate paired-end sequencing using `simNGS` [13] and the runfile shipped with the package source:

  ```
  $ cat frags.fas | simNGS -p paired -o fastq -O reads test/cov/s_4_0066.runfile
  ```

  The files `reads_end1.fq` and `reads_end1.fq` contain the simulated paired-end reads:

  ```
  @Frag_24929 refB (Strand - Offset 11583 -- 12067) 151M
  GCCCCGAGTAGTTCTGGGCGGGGCCCCCGCGGCCAGCGCCGCCCACTATATATTATTTATTCTAACTATT ...
  +
  GGGEFG(EGFGGDGBFEGGG;FD7GEGGDGGA=GGGDG7FFGGGGGFGCAGGGGGFF?GGGFG@GGAGGG ...
  ...
  ```

- Simulate 500,000 fragments, skew normal fragment size distribution with a spike, `after_prim_double` fragmentation method, 15 PCR cycles with the specified efficiency parameters, using 4 cores, verbose mode:

  ```
  $ ./rlsim -n 50000 -d "0.9:sn:(600,50,4,300,2000) + 0.1:n:(700,1,600,2000)" \
  -f after_prim_double -c 15 -eg "(1.0,0.5,0.8)" -el "(0.1,0.7,1.0)" \
  -t 4 -v my_transcripts.fas > frags.fas

  10:09:14.327740 Starting up rlsim using maximum 4 cores.
  10:09:14.327809 Initial random seed: 1352196554
  10:09:14.328244 Number of requested fragments: 50000
  10:09:14.328250 Target fragment length distribution components:
  10:09:14.328275         0.10:n:(700, 1, 600, 2000)
  10:09:14.328280         0.90:sn:(600, 50, 4, 300, 2000)
  10:09:14.369978 Finished sampling target lengths.
  10:09:14.370402 Fragmentation method: "after_prim_double" with parameter 0.
  ```

```
10:09:14.370564 Number of PCR cycles: 15
10:09:14.370593 GC dependent efficiency parameters: (1,0.5,0.8)
10:09:14.370613 Length dependent efficiency parameters: (0.1,0.7,1)
10:09:14.378443 Poly(A) tail distribution components:
10:09:14.378484         1.00:g:(150, 1, 0, 220)
10:09:14.378501 Fragments will be cached to rlsim_gob_5704.
10:09:14.378504 Fragmenting transcripts and amplifying fragments:
10:09:14.381961         0       ENST00000371588$855
10:09:14.393732         1       ENST00000367772$981
10:09:14.425730         2       ENST00000359326$0
10:09:14.425893         3       ENST00000374005$9899
10:09:14.629702         4       ENST00000002165$8381
10:09:14.736127 Initialized 4 transcripts.
10:09:15.474069 Total number of fragments in the pool: 7.29577e+06
10:09:15.474144 Sampling ratio: 0.0068532862192750045
10:09:15.474193 Sampled 50000 fragments.
10:09:15.474211 Missing fragments: 0

$ ../tools/plot_rlsim_report
```

- Estimate parameters from SAM file sorted by read name using `effest` (verbose mode):

```
$ ../tools/effest -v -f ../tools/test/ref.fas ../tools/test/aln1.sam

[12-11-06 10:10:19] Assumed mean efficiency: 0.87
[12-11-06 10:10:19] Number of PCR cycles: 11
[12-11-06 10:10:19] Sliding window step ratio: 10
[12-11-06 10:10:19] Minimum mapping quality: 10
[12-11-06 10:10:19] Saving estimated raw parameters to file: raw_params.json
[12-11-06 10:10:19] Parsing fragments from file: ../tools/test/aln1.sam
[12-11-06 10:10:22] Total number of fragments: 47142
[12-11-06 10:10:22] Total number of fragments from single isoform genes: 47142
[12-11-06 10:10:32] Saving fragment counts to file: effest_counts.pk
[12-11-06 10:10:32] Calculating fragment prior
[12-11-06 10:10:41] Saving fragment prior to file: effest_pr.pk
[12-11-06 10:10:41] Estimated minimum efficiency: 0.513163
[12-11-06 10:10:41] Estimated shape parameter: 5.57905
[12-11-06 10:10:43] Saving estimated expression levels to file: _expr.fas
[12-11-06 10:10:43] Estimating fragment size distribution (truncated normal)
[12-11-06 10:10:43] Estimated mean fragment length: 464
[12-11-06 10:10:43] Estimated fragment length standard deviation: 87
[12-11-06 10:10:44] Truncated normal AIC: 553765
[12-11-06 10:10:44] Estimating fragment size distribution (skew normal)
[12-11-06 10:10:44] Estimated location parameter: 364
[12-11-06 10:10:44] Estimated scale parameter: 132
[12-11-06 10:10:44] Estimated shape parameter: 2.7782
[12-11-06 10:10:45] Skew normal AIC: 552143
[12-11-06 10:10:45] Absolute AIC difference: 1621.73

Suggested rlsim parameters:
        -d "1.0:sn:(364, 132, 2.7782, 229, 868)"
        -eg "(5.57905, 0.513163, 1)"
        -n 47142
```

- Estimate parameters from SAM file sorted by read name using `effest` – assume 15 PCR cycles and a pool efficiency of 0.9:

```
$ ../tools/effest -v -c 15 -m 0.9 -f ../tools/test/ref.fas ../tools/test/aln1.sam

[12-11-06 10:12:14] Assumed mean efficiency: 0.9
[12-11-06 10:12:14] Number of PCR cycles: 15
[12-11-06 10:12:14] Sliding window step ratio: 10
[12-11-06 10:12:14] Minimum mapping quality: 10
[12-11-06 10:12:14] Saving estimated raw parameters to file: raw_params.json
[12-11-06 10:12:14] Parsing fragments from file: ../tools/test/aln1.sam
[12-11-06 10:12:17] Total number of fragments: 47142
[12-11-06 10:12:17] Total number of fragments from single isoform genes: 47142
[12-11-06 10:12:26] Saving fragment counts to file: effest_counts.pk
[12-11-06 10:12:26] Calculating fragment prior
[12-11-06 10:12:35] Saving fragment prior to file: effest_pr.pk
[12-11-06 10:12:35] Estimated minimum efficiency: 0.626195
[12-11-06 10:12:35] Estimated shape parameter: 5.61645
[12-11-06 10:12:37] Saving estimated expression levels to file: _expr.fas
[12-11-06 10:12:37] Estimating fragment size distribution (truncated normal)
[12-11-06 10:12:37] Estimated mean fragment length: 464
[12-11-06 10:12:37] Estimated fragment length standard deviation: 87
[12-11-06 10:12:38] Truncated normal AIC: 553765
[12-11-06 10:12:38] Estimating fragment size distribution (skew normal)
[12-11-06 10:12:38] Estimated location parameter: 364
[12-11-06 10:12:38] Estimated scale parameter: 132
[12-11-06 10:12:38] Estimated shape parameter: 2.7782
[12-11-06 10:12:39] Skew normal AIC: 552143
[12-11-06 10:12:39] Absolute AIC difference: 1621.73

Suggested rlsim parameters:
        -d "1.0:sn:(364, 132, 2.7782, 229, 868)"
        -eg "(5.61645, 0.626195, 1)"
        -n 47142
```

- Simulate 20,000 fragments using the raw parameters estimated by `effest` , set minimum GC-dependent efficiency to 0.5 (verbose mode):

```
$ ./rlsim -v -n 20000 -j raw_params.json -jm 0.5 my_transcripts.fas > frags.fas

10:13:50.240213 Starting up rlsim using maximum 4 cores.
10:13:50.240353 Initial random seed: 1352196830
10:13:50.240887 Using raw parameter file: raw_params.json
10:13:50.241033 Number of requested fragments: 20000
10:13:50.314449 Finished sampling target lengths from raw size distribution.
10:13:50.314969 Fragmentation method: "after_prim_double" with parameter 0.
10:13:50.315094 Number of PCR cycles: 15
10:13:50.315114 Using raw GC efficiencies with a minimum efficiency: 0.5
10:13:50.315956 Poly(A) tail distribution components:
10:13:50.316001         1.00:g:(150, 1, 0, 220)
10:13:50.316043 Fragments will be cached to rlsim_gob_5760.
10:13:50.316058 Fragmenting transcripts and amplifying fragments:
10:13:50.317036         0       ENST00000371588$855
10:13:50.328412         1       ENST00000367772$981
10:13:50.359313         2       ENST00000359326$0
10:13:50.359447         3       ENST00000374005$9899
10:13:50.561491         4       ENST00000002165$8381
10:13:50.672723 Initialized 4 transcripts.
```

```
10:13:50.987213 Total number of fragments in the pool: 1.28970086e+08
10:13:50.987261 Sampling ratio: 0.00015507472019519317
10:13:50.987331 Sampled 20000 fragments.
10:13:50.987386 Missing fragments: 0

$ ../tools/plot_rlsim_report
```

2 Simulating RNA-seq library construction using `rlsim`

2.1 Quick reference

```
Simulate RNA-seq library preparation with priming biases, PCR biases and size selection (version: 1.

Usage:
        rlsim [arguments] [transcriptome fasta files (optional)]

Optional arguments:
                argument                    type    default
        -n      requested fragments         int
        -d      fragment size distribution  string  [check source]
        -f      fragmentation method        string  "after_prim_double"
        -b      strand bias                 float   0.5
        -c      PCR cycles                  int     11
        -p      priming bias parameter      float   5.0
        -k      primer length               int     6
        -a      poly(A) tail size dist.     string  [check source]
        -flg    fragment loss probability   float   0.0
        -m      expression level multiplier float   1.0
        -e      fixed PCR efficiency        float   0.0
        -eg     GC efficiency parameters
                as "(shape, min, max)":
                    shape                   float   8.0
                    min                     float   0.8
                    max                     float   0.95
        -el     length efficiency parameters
                as "(shape, min, max)":
                    shape                   float   0.0
                    min                     float   1.0
                    max                     float   1.0
        -j      raw parameter file          string
                superseeds -d, -c, -eg
        -jm     minimum raw gc efficiency   float   0.0
        -r      report file                 string  "rlsim_report.json"
        -t      number of cores to use      int     4
        -g      keep fragments in memory    bool    false
        -si     initial random seed         int     from UTC time
        -sp     pcr random seed             int     auto
        -ss     sampling random seed        int     auto
        -gobdir fragment directory          string  "rlsim_gob_$PID"
        -v      toggle verbose mode         bool    false
        -h      print usage and exit        bool    false
        -V      print version and exit      bool    false
        -prof   write CPU profiling info    string  ""
        -gcfreq trigger garbage collection  int     100
```

```
                after this many transcripts
        -randt  generate RNG test files     bool    false

Examples:
        rlsim -n 2000000 transcripts.fa
        cat transcripts.fa | rlsim -n 2000000

For more details consult the package manual at:
        https://github.com/sbotond/rlsim/tree/master/doc/rlsim_manual.pdf
```

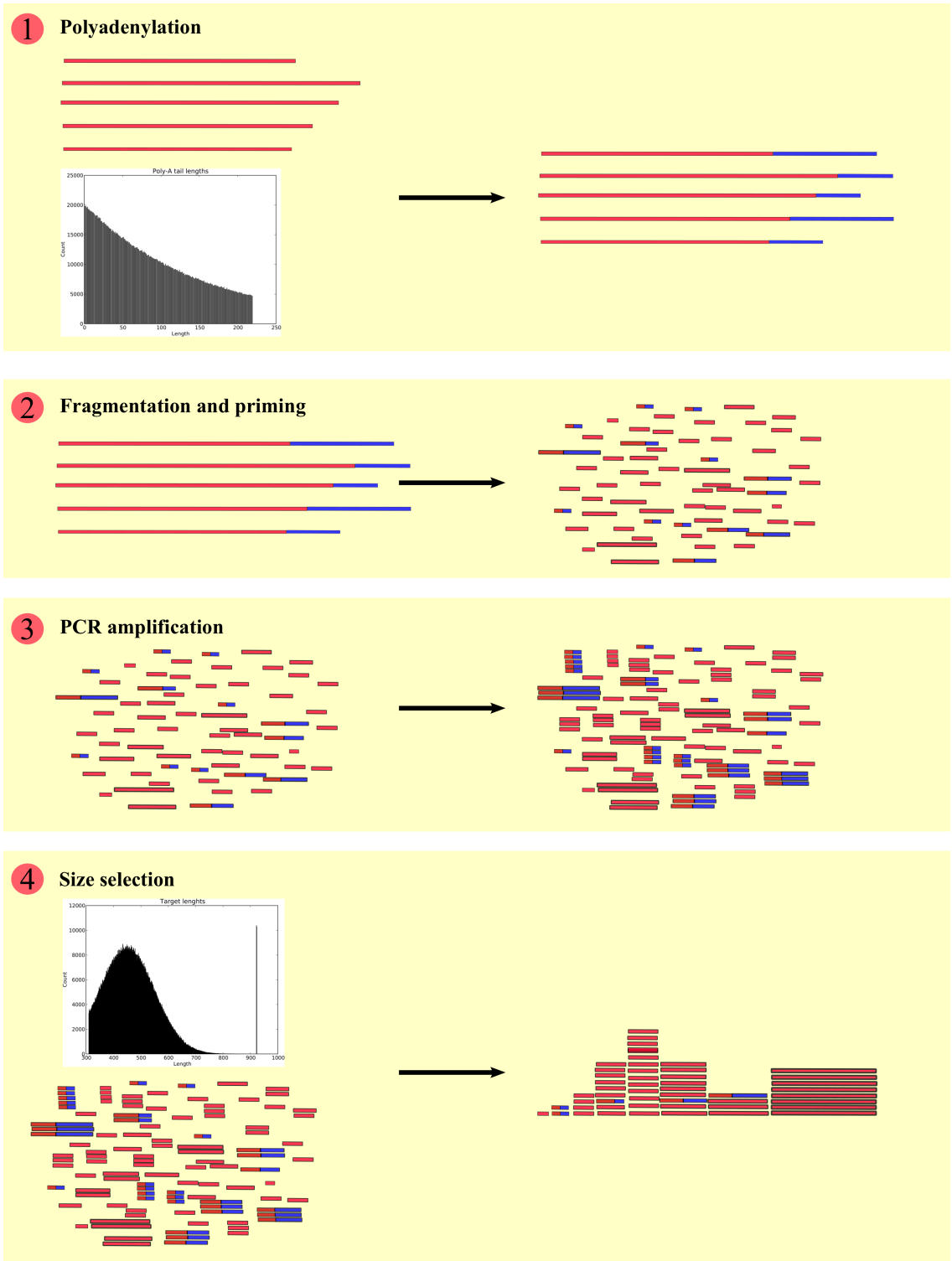
2.2 Background

2.2.1 An overview of the simulation framework

Conceptually, the simulation of RNA-seq library construction consists the following main steps:

1. Sample the fragment lengths specified by the target size distribution (analytical mixture or empirical distribution).
2. For every molecule belonging to a particular transcript:
   a) Simulate polyadenylation.
   b) Simulate fragmentation and (optionally) priming.
3. For every individual fragment in the pool generated by the fragmentation process, simulate PCR amplification with efficiencies dependent on fragment size and GC-content.
4. Simulate size selection by sampling from the pool fragments with lengths obtained in the first step.
5. For every fragment sample the strand of origin and report it Fasta/`simLibrary` [13] format.

   The main steps are illustrated in the figure below:



Notes:

- The fragment sampling (one of the most expensive steps) is performed in parallel using the concurrency capabilities of the `Go` language.
- During RNA-seq library preparations, size selection is usually performed before PCR amplification. In `rlsim` , for practical reasons, size selection is simulated after PCR amplification. In the majority of the cases this should not affect considerably the simulation of the interactions between size selection and other biasing factors.

2.2.2 Target size distributions

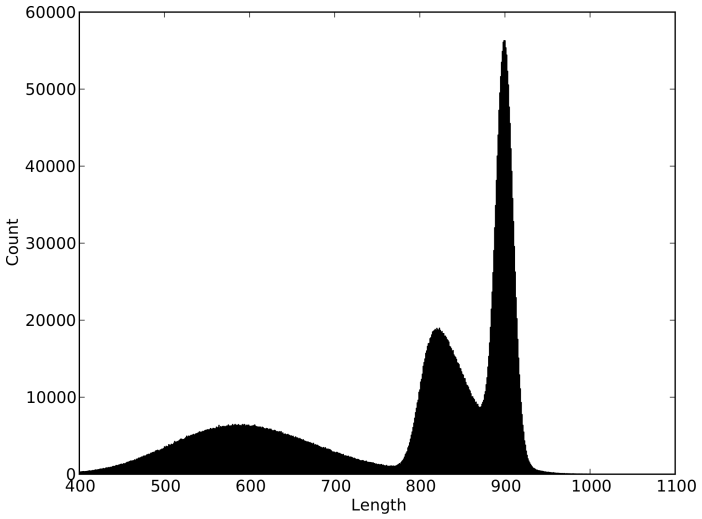
The target length distributions used for the simulation of size selection and polyadenylation can be specified as mixture distributions with the syntax *weight*:*type*:*parameters*. The available types are the following:

- `n` – truncated **normal** distribution [17] parametrised as (*mean*, *sd*, *minimum*, *maximum*). For example a truncated normal mixture component with mean 900, standard deviation 10 with minimum value 100 and maximum value 2000 can be specified as `n:(900, 10, 100, 2000)`.
- `sn` – truncated **skew normal** distribution [18] parametrised as (*location*, *scale*, *shape*, *minimum*, *maximum*) A skew normal component with location 800, scale 50, shape 4 truncated between 600 and 1000 is specified as `sn:(700,50,4,600,1000)`.
- `g` – truncated **gamma** distribution [16] parametrised as (*mean*, *shape*, *minimum*, *maximum*). A gamma distribution with mean 600, shape 50, minimum value 400 and maximum value 1000 is specified as `g:(600, 50, 400, 1000)`.

The mixture weight are normalised in order to sum to 1, hence an equal mixture of the example distributions above can be specified as:

```
1.0:n:(900, 10, 100, 2000) + 1.0:sn:(800,50,4,600,1000) + 1.0:g:(600,50, 400, 1000)
```

, having a density similar to the one shown in the figure below:

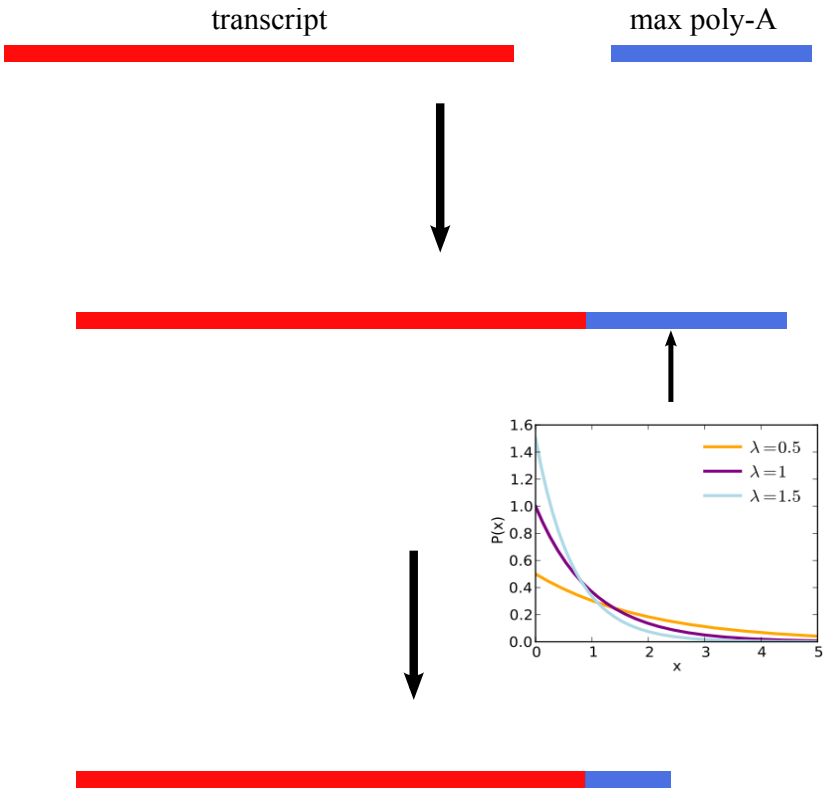


2.2.3 Simulating polyadenylation

The target distribution of poly(A) tail lengths is specified as a mixture of truncated distributions via the `-a` flag using the syntax described in section 2.2.2. Polyadenylation is simulated in the following steps:

- During the processing of the input, the poly(A) tail with the maximum size (as inferred from the target distribution) is appended to the sequence of each transcript.
- During the fragmentation of a given molecule, a length is sampled from the target distribution and the poly(A) tail is shortened to that length.

The simulation of polyadenylation is illustrated in the figure below:



2.2.4 A note on simulating TSS and poly(A) site variability

`rlsim` uses the transcriptome as input, hence it cannot simulate the variability of polyadenylation and transcription start sites. During the simulation it is assumed that the transcription starts at the first base of the transcript sequence and that the poly(A) tail is appended after the last one. However, the user can still simulate these factors by simply including the TSS and poly(A) variants in the input fasta file as distinct transcripts.

2.2.5 Simulating priming

Some of the fragmentation methods simulate priming based on the "binding affinity" of the first `-k` bases (6 by default) of the proposed fragment. The priming site is sampled using the following approach:

- Pre-calculate and cache the binding affinity of all sites in the transcript. The first step in calculating the binding affinity is to calculate the change in free energy ΔG corresponding to the binding of the oligomer beginning with the respective base, based on a nearest-neighbor thermodynamic model [28] with the "temperature" equal to the priming bias parameter p (set by the `-p` flag). Then the binding affinity K is calculated as:

$$K = e^{-\frac{\Delta G}{Rp}} \tag{1}$$

  , where R is the universal gas constant.

- During the simulation of fragmentation, the priming site within individual fragments is sampled proportionally to the pre-calculated binding affinities. The sampled priming site will define the new start or end position of the fragment.

  WARNING: The above approach for simulating priming bias is inspired by the physical model of oligonucleotide binding, but it ignores many important factors and so it is not a valid physical model itself! For example, it assumes that every fragment is primed once and only once, furthermore it ignores the effect of the concentrations of random oligomers. Hence, p is a mere "priming bias parameter" and not real temperature on the Kelvin scale. Despite these caveats, we think that this

approach is still useful for creating complex priming biases which are qualitatively similar to the ones observed in real data.

Under positive priming bias parameters (p), the approach above will create a preference for G and C nucleotides, especially at the first and last bases of the oligomer. Negative priming parameters invert this, creating preference towards A and T nucleotides.

2.2.6 Fragmentation methods

`rlsim` has multiple fragmentation methods, some of them are inspired by actual protocols:

- `after_prim_double - default`
- `after_noprim_double`
- `pre_prim`
- `after_prim`
- `after_noprim`
- `pre_prim`
- `prim_jump`

The fragmentation methods are specified via the `-f` flag and optionally they can take a parameter separated with a "`:`" from the name of the approach (e.g. `"pre_prim:2000"`).

In the section below we will briefly describe the fragmentation methods, their parameters and the characteristics of the data simulated using them. In order to illustrate the different fragmentation/priming methods, we will simulate data under two rather artificial settings in order to emphasize their effects on the sequence biases around the edges of the fragment and the sequence dependent coverage trends across the transcript.

1. The goal of the first simulation setting is to illustrate the **sequence specific biases** around the fragment edges. In order to clearly demonstrate these effects, we simulated fragments from a rather unrealistic random transcriptome composed of 500 transcripts of length 1000000 having uniform base composition and an expression level of 100. The length of the transcripts ensures that the effect of the fragmentation biases is minimised. The large number of random bases in the transcriptome ensures that most of the oligomers are represented in the input in an equal frequency. The other relevant parameters were the following:

   - The number of sampled fragments (`-n`): 5000000
   - The target insert size distribution was a truncated normal with a mean of 200 and a standard deviation of 20: `"1.0:n:(200, 20, 150, 100000)"`
   - The priming bias parameter (`-p`) was set to 5.0.
   - The length of the simulated poly(A) tail (`-a`) was set to 1, unless indicated otherwise.
   - In most simulations the simulation of PCR amplification was disabled by setting the number of cycles to 0.
   - In the cases where PCR amplification was simulated, we used 11 cycles with the following GC efficiency parameters: `"(0.2, 0.5, 0.95)"`
   - Sequencing was simulated using `simNGS`, with the read length set to 75.
   - The simulated reads were aligned back to the transcriptome using `BWA` in paired-end mode.
   - The sequence bias plots were produced using the `pb_plot` tool.

   The simulation pipeline and the raw output can be found under the `src/test/pos_bias` directory in the `rlsim` repository.

2. The second simulation setting illustrates how the sequence specific and fragmentation biases manifest themselves as **uneven coverage across the the transcripts**. In order to illustrate this, we simulated from a transcript with a total length of 3500, composed of alternating GC rich and AT rich units of length 35, having an expression level of 400000. Other relevant simulation parameters:

   - The number of requested fragments (`-n`) was 40000.
   - The target insert size distribution (`-d`) was a truncated normal with a mean of 200 and a standard deviation of 20: `"1.0:n:(200, 20, 150, 20000)"`
   - The priming bias parameter (`-p`) was set to 1.0, producing very strong sequence biases.
   - The strand bias parameter (`-b`) was set to 0.0, corresponding to generating a "stranded" library.
   - The length of the simulated poly(A) tail (`-a`) was set to 1.
   - In most simulations the simulation of PCR amplification was disabled by setting the number of cycles to 0.
   - In the cases where PCR amplification was simulated, we used 30 cycles with the following GC efficiency parameters: `"(8.0,0.2,0.95)"`
   - Sequencing was simulated using `simNGS`, with the read length set to 50.
   - The simulated reads were aligned back to the transcript using `BWA` in paired-end mode.
   - The sequence bias plots were produced using the `plot_cov` tool, which plots the *base coverage* of individual reference bases using the colors blue for A and T, and red for G and C.

   The simulation pipeline and the raw output can be found under the `src/test/cov` directory in the `rlsim` repository.

WARNING: Please note that the simulations above were deliberately designed to emphasize the effects of different fragmentation/priming methods. When simulating from real transcriptomes and more reasonable parameters these effects will combine in a complex manner, making them less apparent and harder to link them to specific features in the coverage.

Given the lack of sufficient of knowledge about the RNA fragmentation process *in general*, the fragmentation methods implemented in `rlsim` do not apply sophisticated approaches for the RNA fragmentation itself, but they try to produce a fragment pool with a length distribution similar to the specified target distribution. This is generally done by trying to match the average fragment lengths produced by fragmentation to the mean lengths of the mixture components.

Fragmentation method: `after_prim_double`

This is the default fragmentation method, inspired by the standard, non-stranded Illumina RNA-seq protocol. It is meant to introduce sequence specific "priming" biases near the start and end of the fragment. Fragmentation is simulated on the level of individual transcript "molecules", meaning that if a transcript has an expression level of 100, than fragmentation will be simulated a hundred times, each fragment produced during these simulation is registered in a "fragment pool" which later is subject to PCR simulation and size selection.

The fragmentation of a single transcript is done in following steps, as illustrated in the figure below:

1. As a first step, a mixture component is sampled from the target mixture specified by the user.

2. The number of breakpoints is sampled from a Poisson distribution such that the average length of produced fragments will be equal to *twice* the mean of the mixture component sampled in the first step. The reason for doubling the average fragment length is to compensate for the shortening effects of the priming simulation performed later on. Alternatively the mean fragment length can be specified as a parameter passed to the fragmentation method (e.g. `"after_prim_double:600"`)

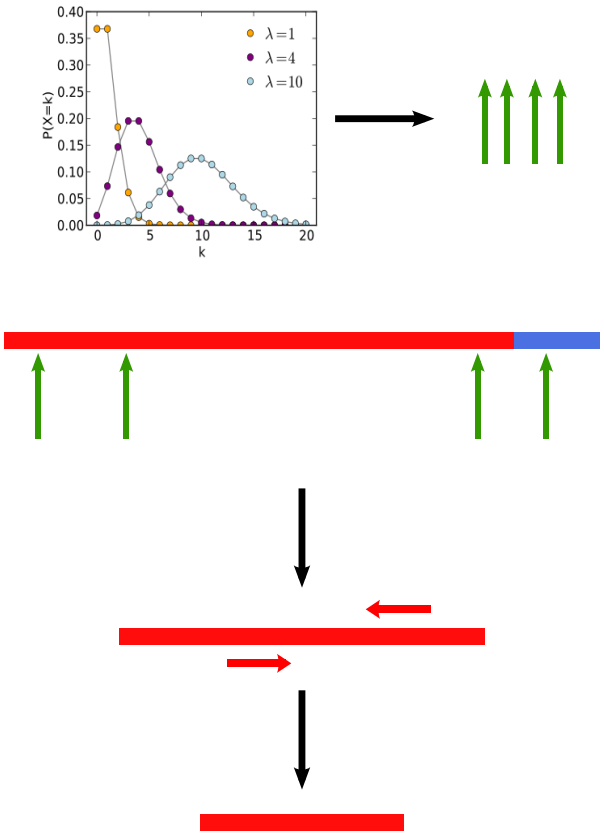
3. The breakpoints are sampled *uniformly* across the length of the transcript. The beginning of the transcript and the end of the poly(A) serve as pre-defined "extra" breakpoints.

4. All the fragments which are outside of the range of the specified target mixture are discarded right away.

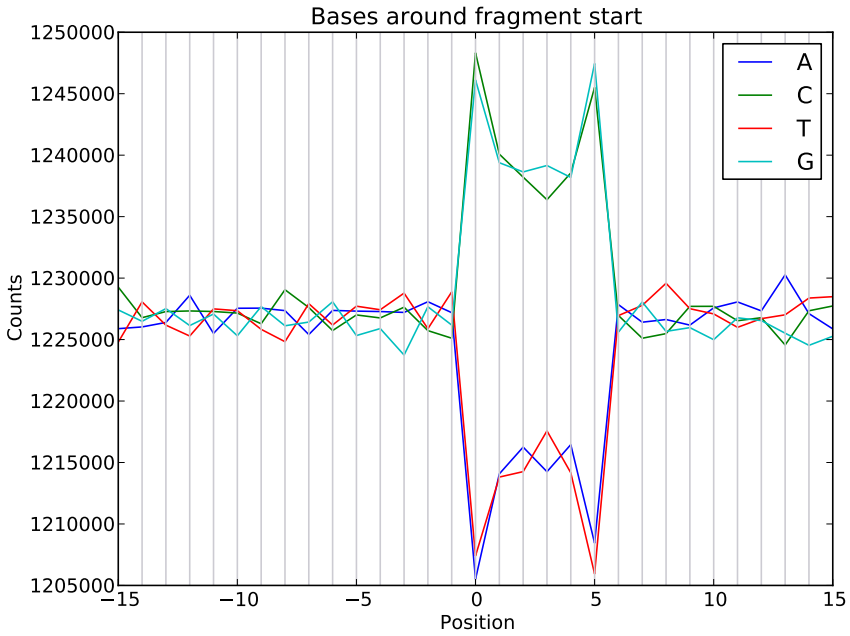
5. Priming is simulated on the forward and reverse strands as described in section 2.2.5 in order to select a new fragment start and end.

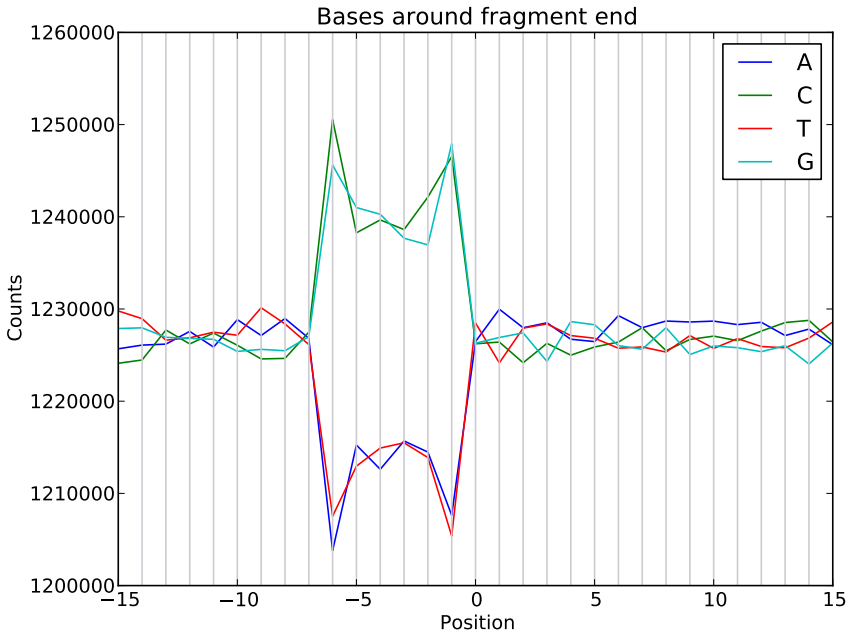
6. The fragment is discarded with a probability set by the `-flg` flag, which allows for the tuning of the total number of fragments produced from a fixed input transcriptome.

7. Finally, the fragment is registered in the fragment pool which later is subject to PCR simulation and size selection.

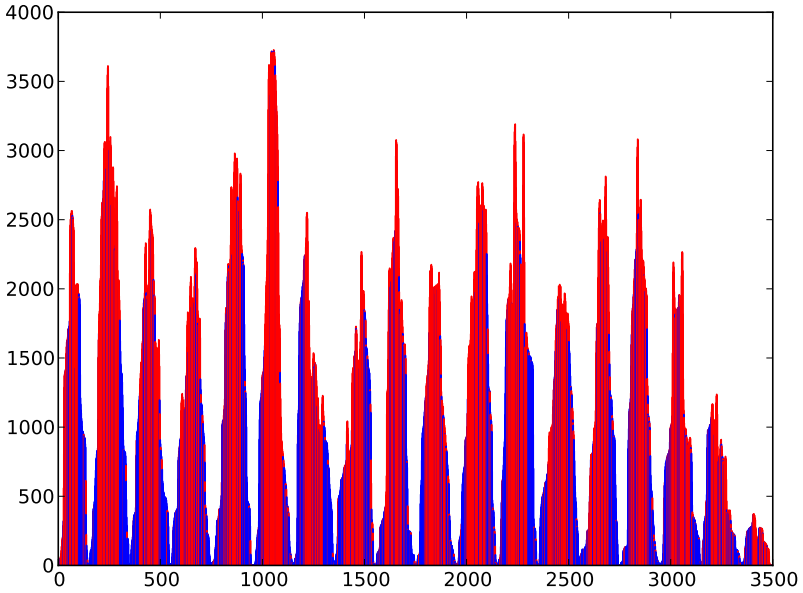


From the output of the first simulation setting using `after_prim_double` it is apparent that this method introduces sequence specific biases both near the the fragment start and end. The priming simulation prefers G and C nucleotides leading to higher binding affinities, especially at the first and last bases of the oligomer:



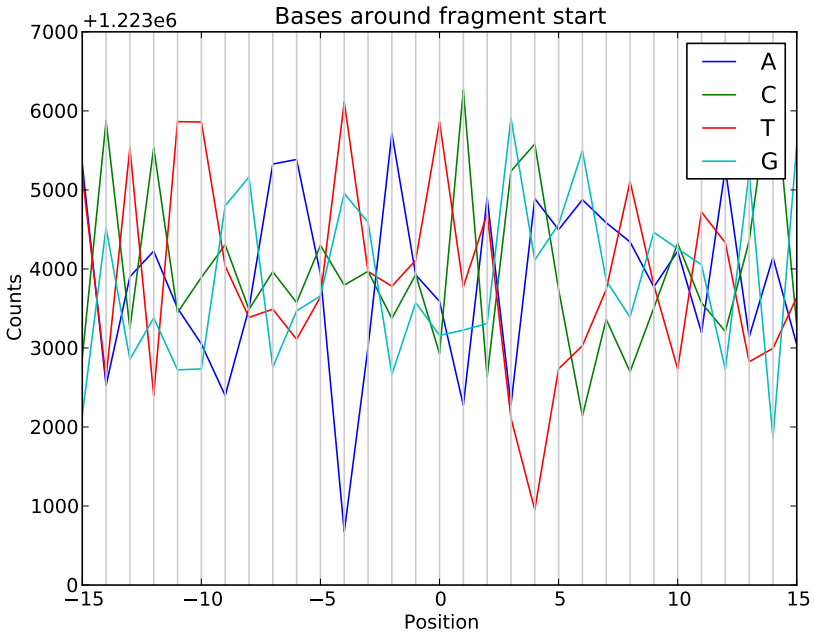


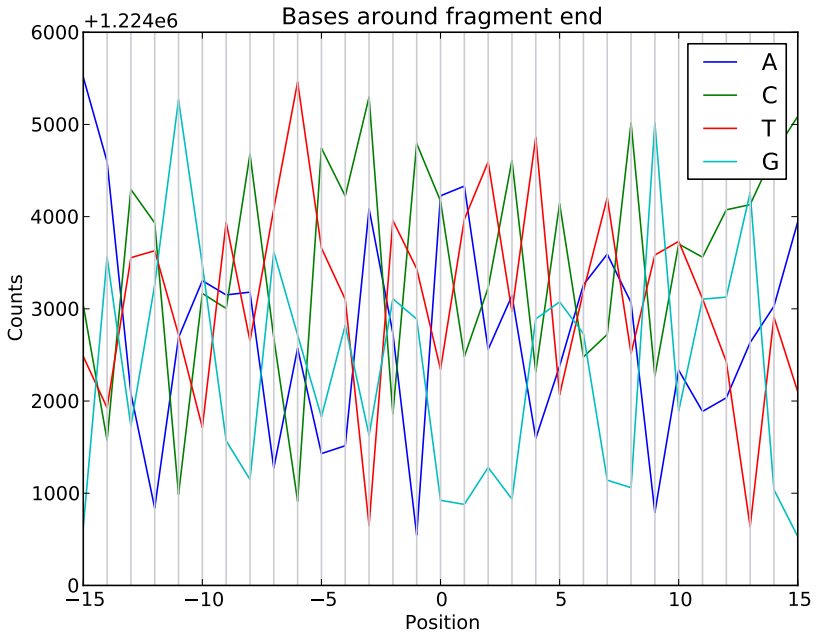
The preference towards G and C nucleotides is also manifested as an unequal coverage across the example transcript in the second simulation, leading to higher base coverage in GC-rich regions:



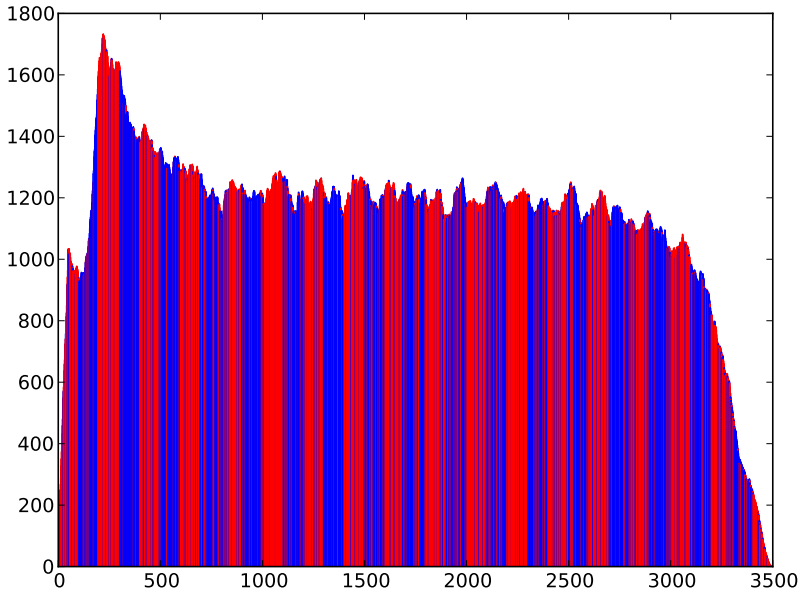
Fragmentation method: `after_noprim_double`

This fragmentation method is very similar to the `after_prim_double` method described above, with one important difference: the priming sites are sampled uniformly, without making use of the binding affinities. Hence this method produces a fragment pool similar to the previous method, but without introducing sequence specific biases. This is apparent from the output of the first simulation setting, showing an essentially random distribution of nucleotides around fragment start and end:



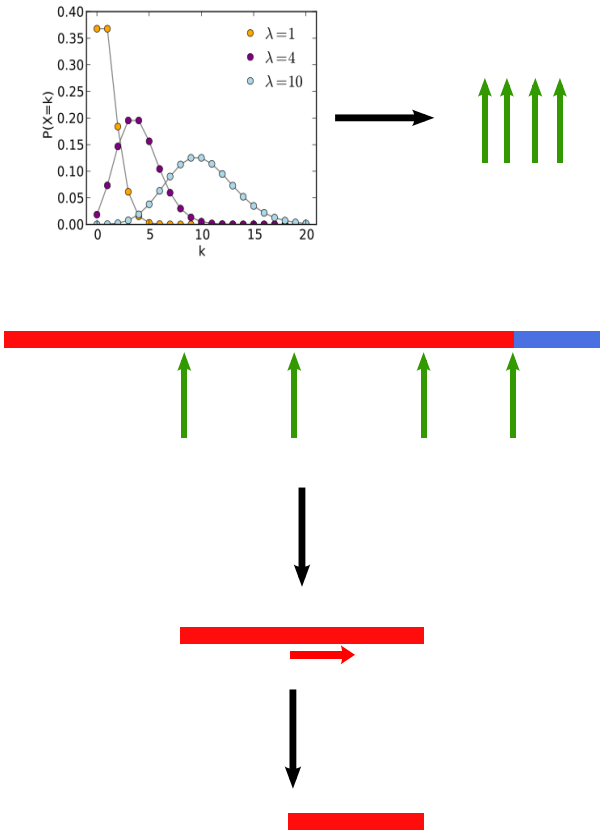


Also, the output of the second simulation setting shows how the base coverage is no longer correlated with the GC content, however we can still observe the characteristic coverage biases around the 3' and 5' of the transcript due to fragmentation, size selection and strandedness:

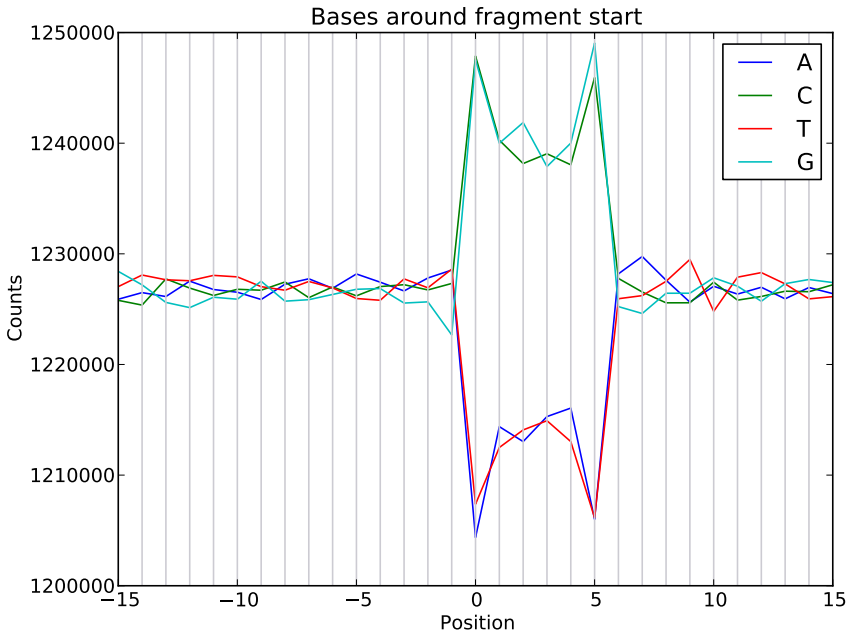


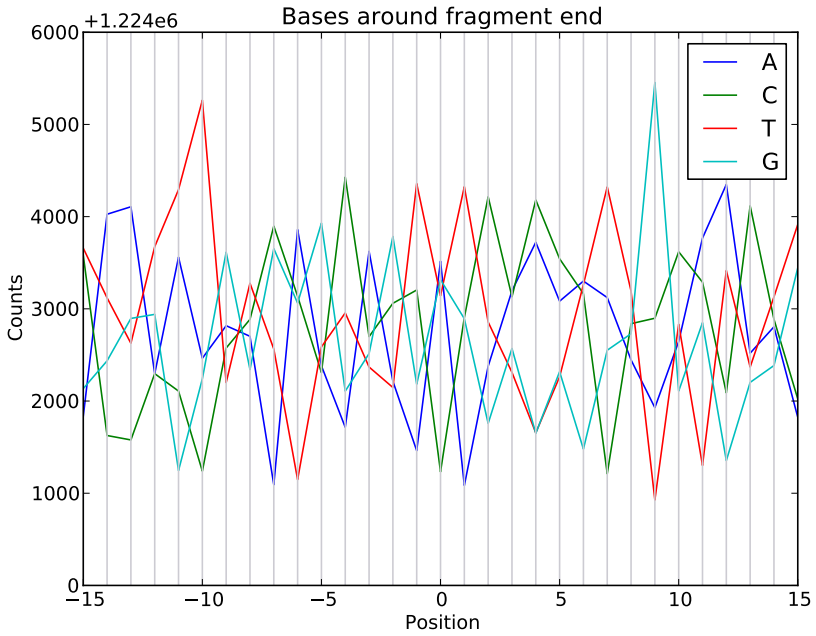
Fragmentation method: `after_prim`

This fragmentation method is similar to `after_prim_double`, however the binding affinities are used only when sampling the start of the fragment.

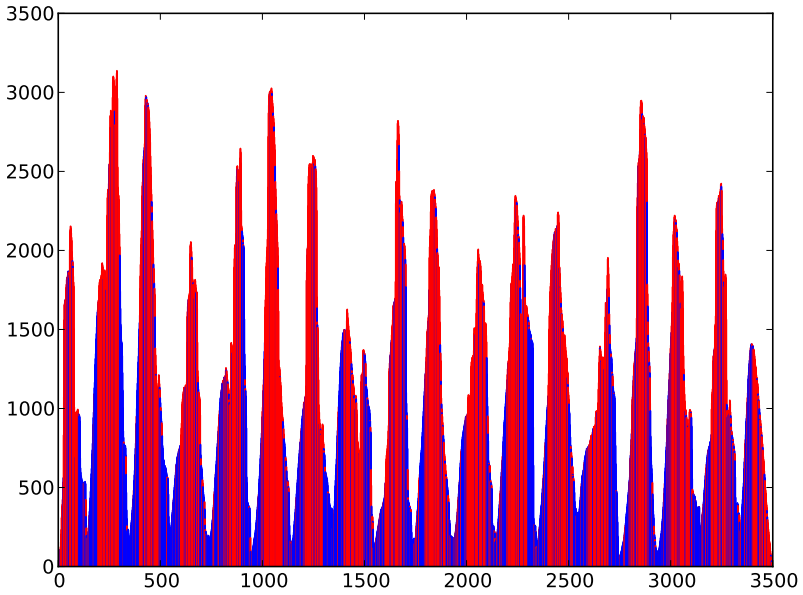


This leads to a sequence specific bias near the start of the fragment, but a random distribution of nucleotides around the end:



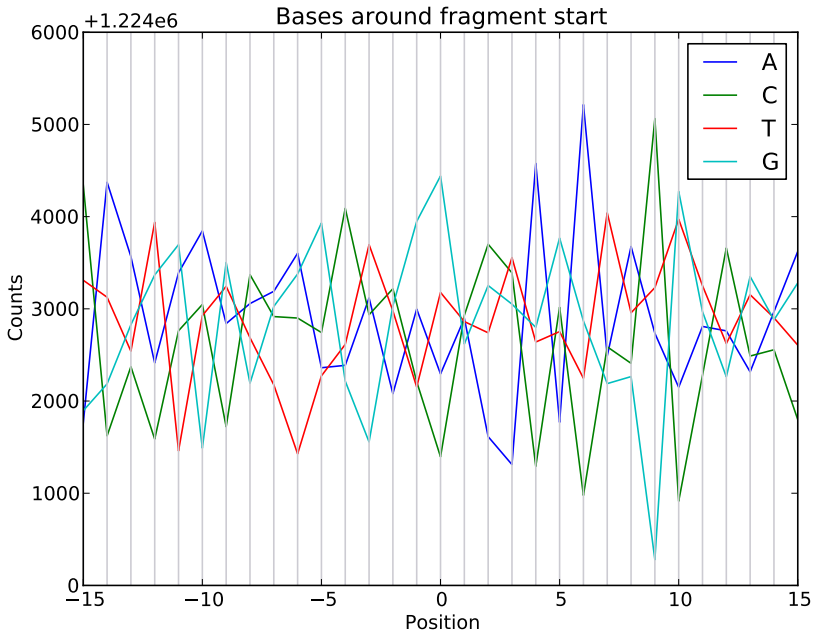


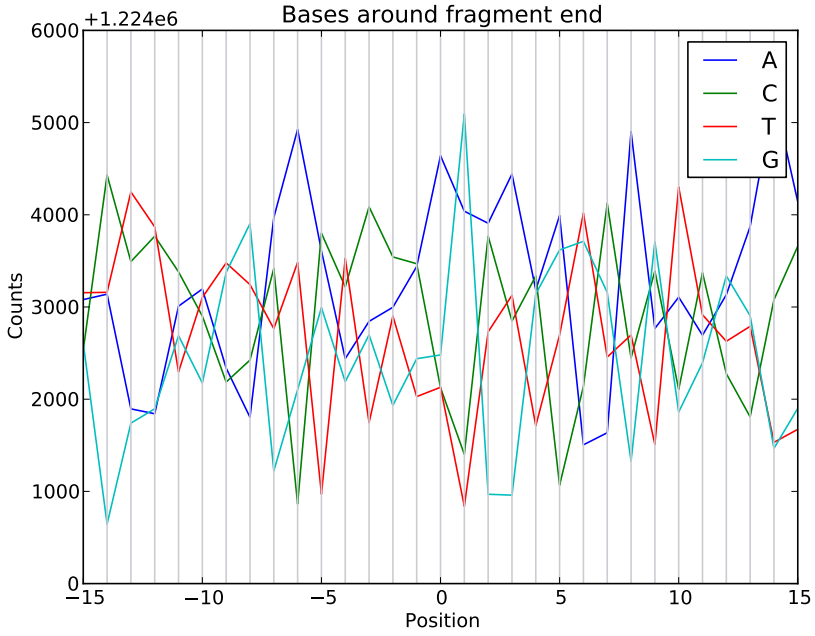
This method, similarly to `after_prim_double` increases coverage in GC rich regions:



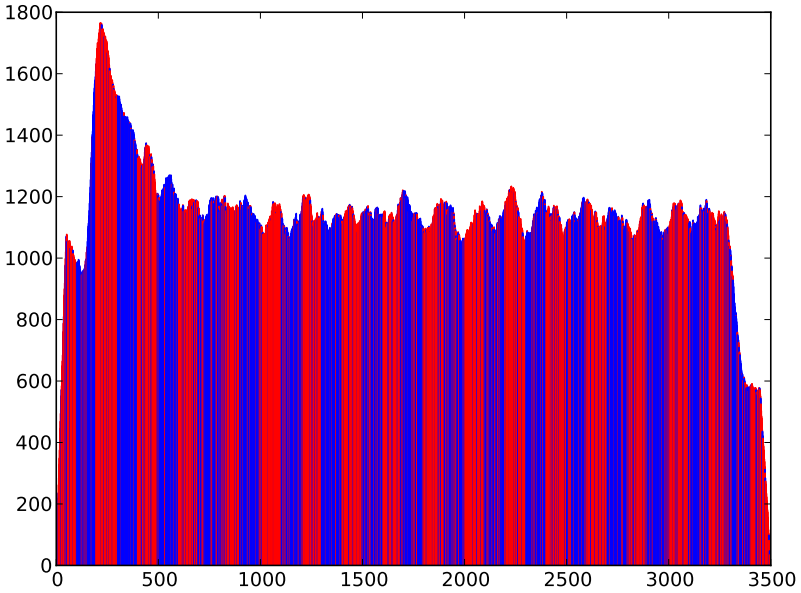
Fragmentation method: `after_noprim`

This fragmentation method is a modification of `after_prim`, with uniformly sampled priming sites. It produces no sequence specific biases near fragment start and end:





The coverage trends due to fragmentation, size selection and strandedness are still apparent:



Fragmentation method: `pre_prim`

This fragmentation method is inspired by the RNA-seq protocols which perform reverse transcription primed in the poly(A) tail. The fragmentation consists of the following steps:

1. The method does *not* simulate the "sliding" of the poly-T primer over the poly(A) tail but it assumes that priming always happens right at the end. However, the subsequent fragmentation step introduces similar effects to "primer sliding".

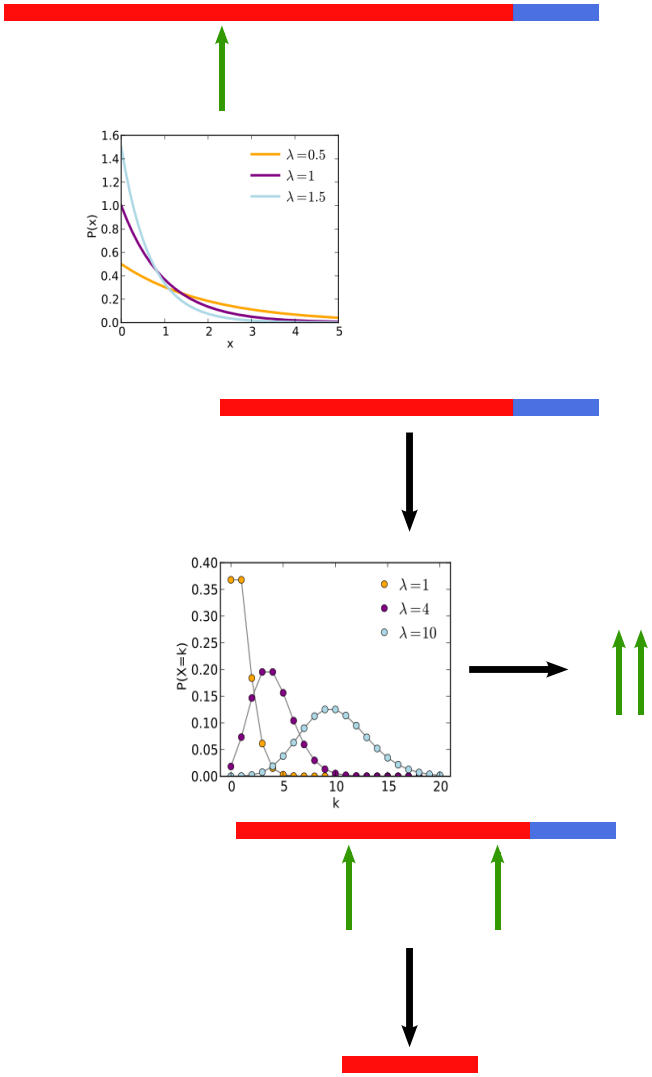
2. Elongation is simulated by sampling the cDNA length from a geometric distribution with a mean specified by the mandatory parameter taken by the fragmentation method (e.g. `"pre_prim:1500"`). The sampled elongation length defines the new start of the transcript.

3. Breakpoints are sampled similarly to the `after_noprim_double` method, without doubling the average length of the produced fragments.

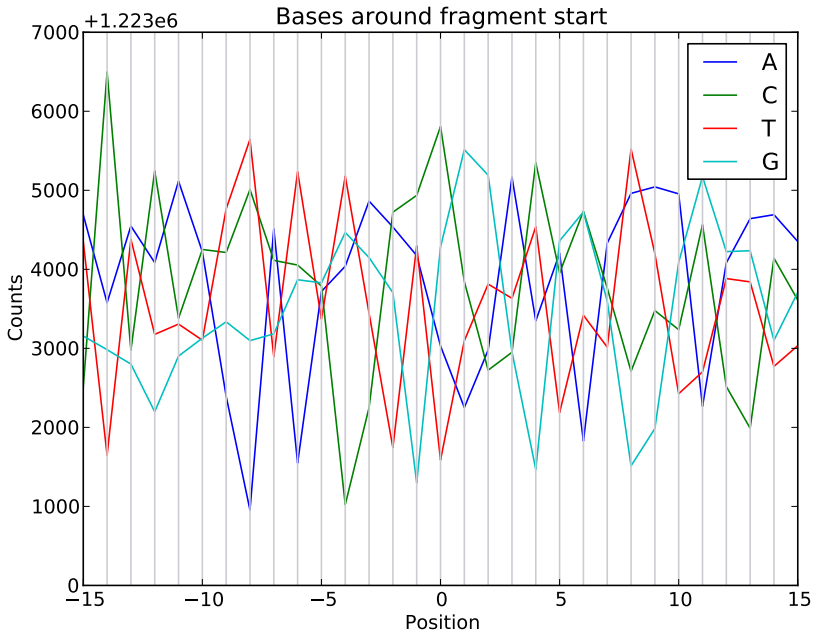
4. Fragments with lengths outside the target mixture are discarded.

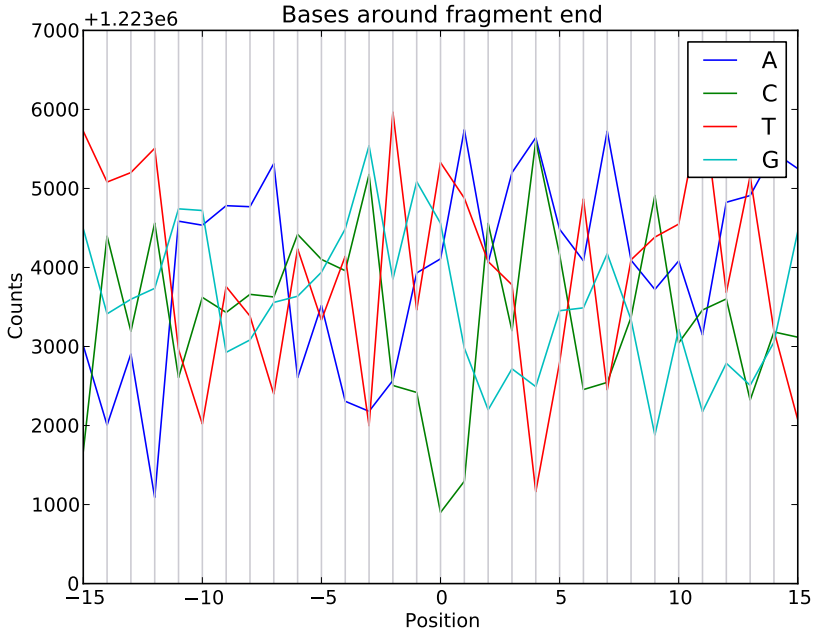
5. Fragment loss is simulated according to the probability specified by the `-flg` flag.

6. The remaining fragments are registered in the fragment pool.

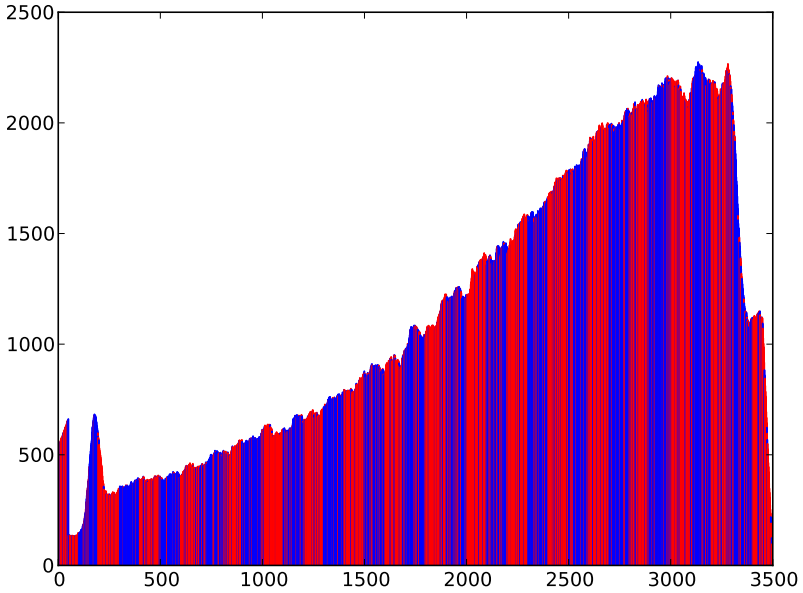


This fragmentation method does not introduce sequence specific biases in an explicit manner, as it is apparent from the output of the first simulation:





This method explicitly introduces a coverage bias towards the 5' end of the transcript, the strength of which depends on the specified mean elongation length parameter:

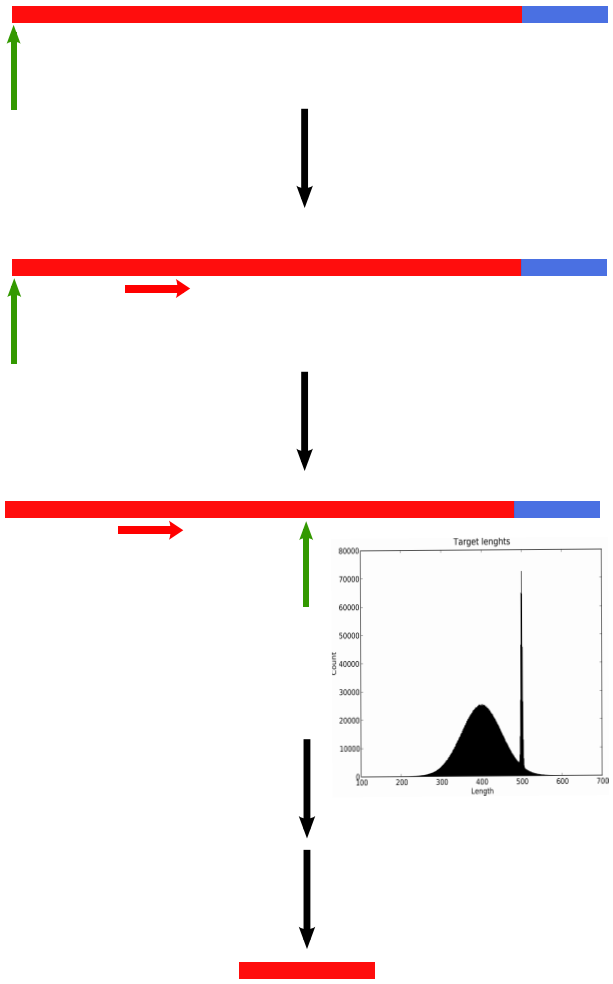


WARNING: This fragmentation method takes the mean elongation length as a mandatory parameter. It is important to carefully choose this parameter as with the default value the simulation might not produce the desired output.

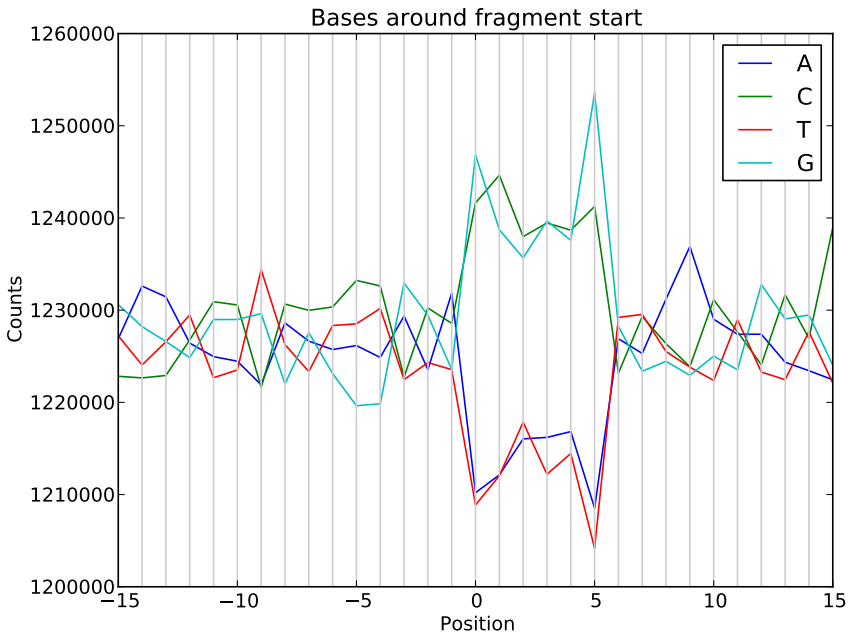
Fragmentation method: `prim_jump`

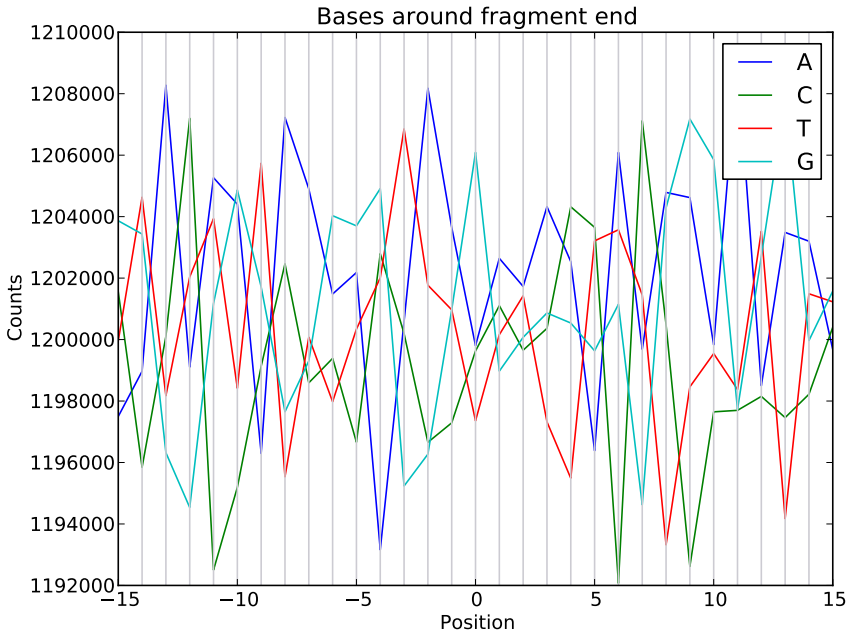
This fragmentation method has no biological inspiration, however it is able to introduce both sequence specific biases and 5' coverage biases in a complex manner, providing an opportunity to stress-test advanced bias correction methods. The following steps are performed:

1. The start of the fragment is sampled by simulating priming across the whole length of the transcript.
2. The end of the fragment is defined by sampling the fragment length from the target length distribution.
3. After simulation of loss, the fragment is registered in the fragment pool.
4. The steps above are iterated over the remaining portion of the transcript until the end of the poly(A) tail is reached.

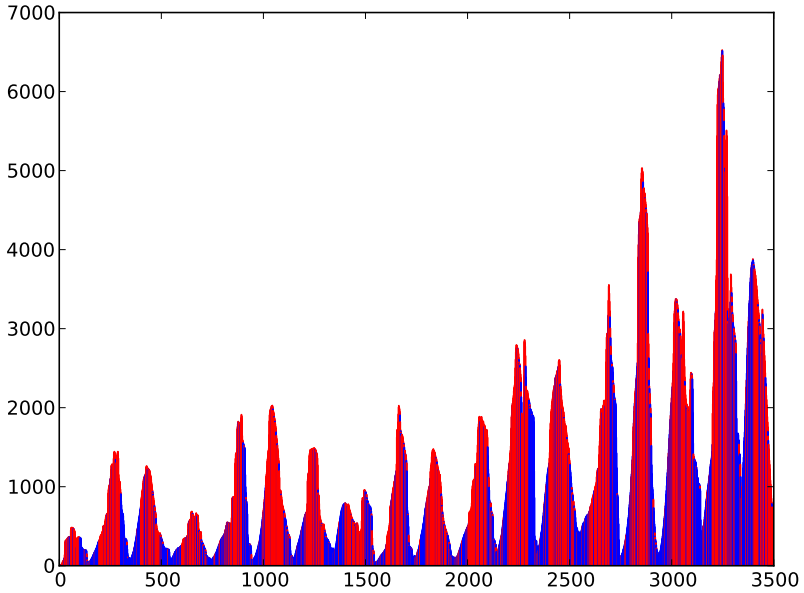


This method introduces sequence specific biases near the start, but not the end of the fragment:





Besides the preference towards CG rich regions, it also introduces a coverage bias towards the 5' end of the transcript, as demonstrated by the output of the second simulation setting:



2.2.7 Simulating PCR amplification

A central concept in the `rlsim` PCR simulation framework is the "amplification efficiency"[2], which is the probability that a fragment is replicated during a cycle. In the `rlsim` framework the amplification efficiency can be fragment specific (as described below), but it is always constant throughout the cycles.

[2]WARNING: The term "efficiency" (denoted as ϵ) is used through this document as the probability of replication, in contrast with the quantitative PCR literature, in which efficiency usually means the expected number of descendants of a single molecule after a single cycle (equal to $1 + \epsilon$).

The PCR simulation takes the fragment pool generated by the fragmentation process (containing fragments with initial count C_0) and outputs a fragment pool with amplified fragment counts. If C_i is the count for a given fragment and ϵ is the amplification efficiency, the amplified count after one cycle (C_{i+1}) is calculated as:

$$C_{i+1} = C_i + Binomial(n = c_i, p = \epsilon) \tag{2}$$

This step is repeated until the number of PCR cycles is equal to the value of the `-c` flag.
Amplification efficiencies can be specified in many ways:

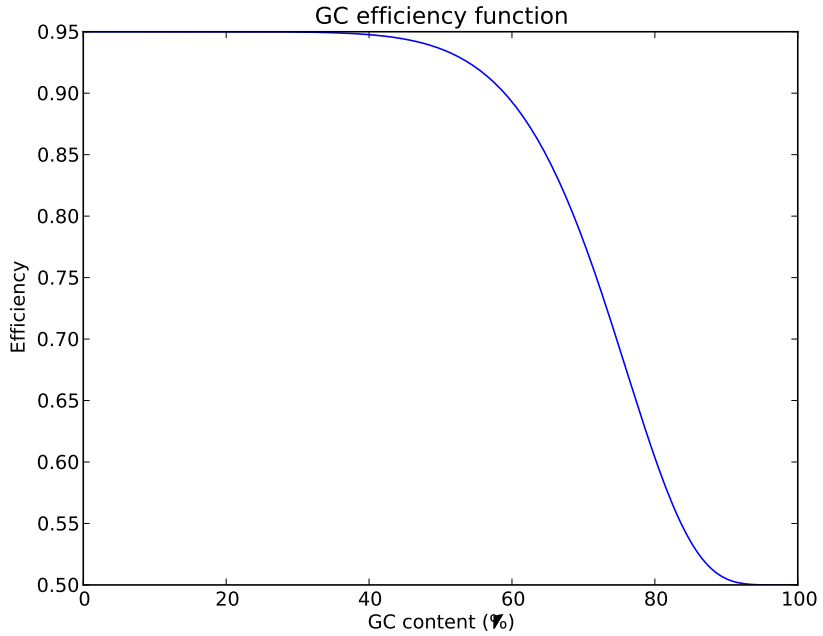
- **Fixed amplification efficiency**: if the `-e` flag is set, its value will be used as a amplification efficiency *for all fragments*, regardless of length and GC content.
- Alternatively the GC dependent efficiency can be set through the `-eg` flag and the length efficiency through the -el. The actual amplification efficiency is calculated as the product of the GC and length efficiencies.
- The "raw parameter" JSON files generated by `effest` and provided by the `-j` flag superseeds the `-eg` flag, and so `rlsim` will use the respective empirical GC efficiencies. A minimum GC efficiency can be specified through the `-jm` flag.

Specifying GC dependent efficiencies

The `-eg` flag accepts strings formatted as "$(\alpha,\ m,\ M)$", where m is the minimum efficiency, M is the maximum efficiency and α is a shape parameter. Then the amplification efficiency of a fragment with GC content g_i is calculated as:

$$\epsilon_{g_i} = m + (M - m)\,(1 - g_i^{\alpha})^{\alpha} \tag{3}$$

For example `"(8.0,0.5,0.95)"` defines the following mapping between GC content and amplification efficiency:



WARNING: The `-eg` flag provides an easy way to specify GC dependent efficiencies, but it can be too restrictive. The user can specify however an arbitrary discrete GC efficiency function by tweaking a "raw parameter" file generated by the `effest` tool.

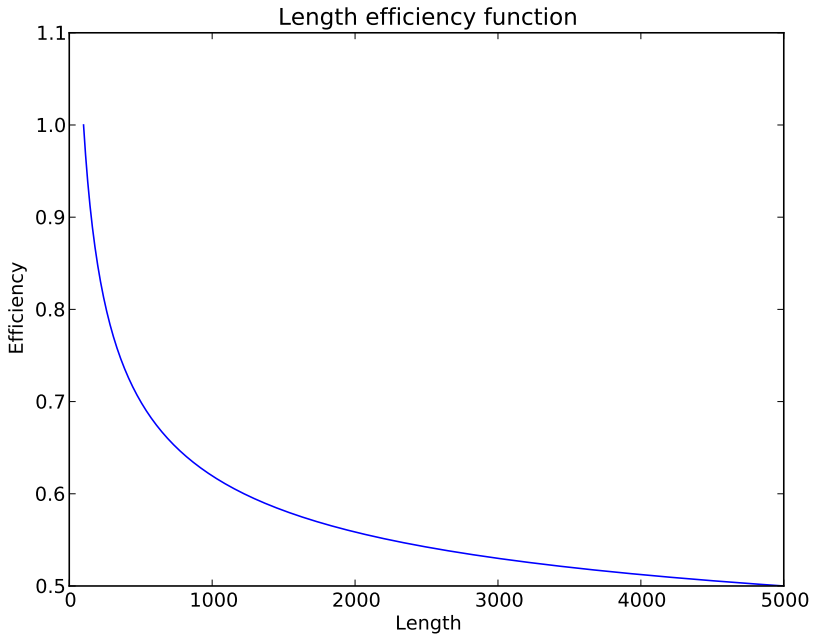
Specifying length dependent efficiencies

The length dependent efficiencies are specified through the `-el` flag as a string formatted as "(β, m, M)", where m and M are the minimum and maximum efficiencies and β is a shape parameter. The efficiencies are calculated as:

$$\epsilon_{l_i} = l_i^{-\beta} \tag{4}$$

, and then scaled in order to have the minimum m and maximum M over the range of the target size distribution specified via the `-d` flag.

For example, if the target size distribution is `"1:n:(300,20,100,5000)"` and the length efficiency parameter is `"(0.4,0.5,1.0)"`, the mapping between lengths and efficiencies is:



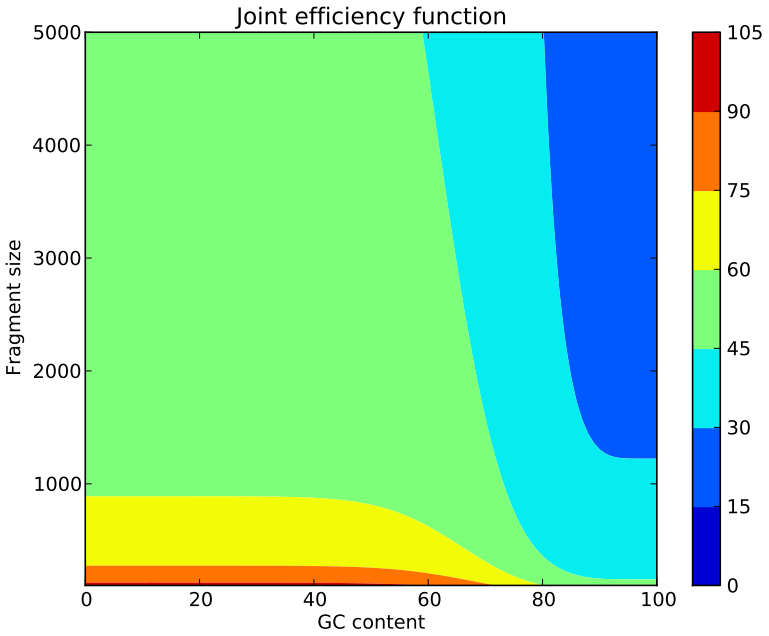
WARNING: Note than the length dependent efficiencies are set to 1.0 by default.

Calculating the joint efficiency function

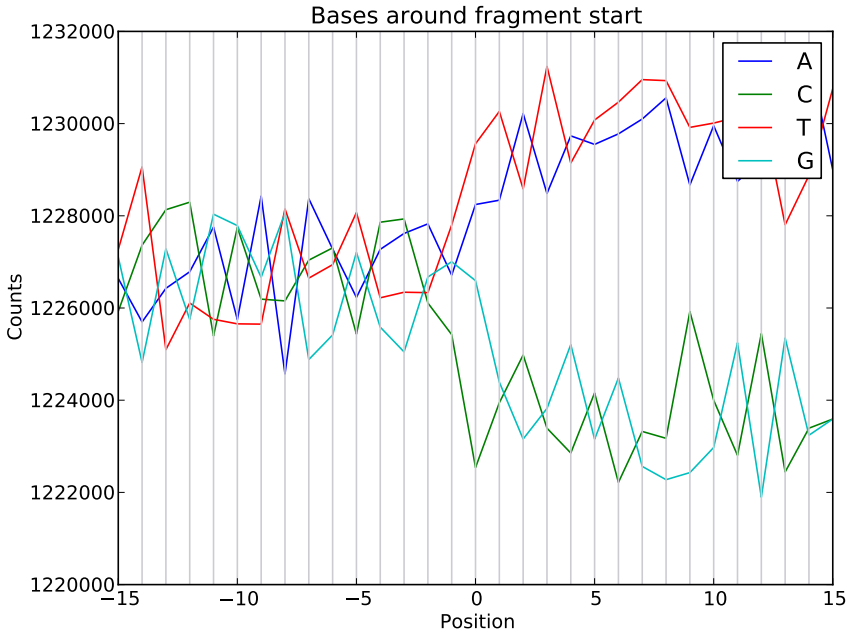
The "joint efficiency function" determining the amplification efficiency of a fragment with a given GC content and length is then calculated as the product of the two components:

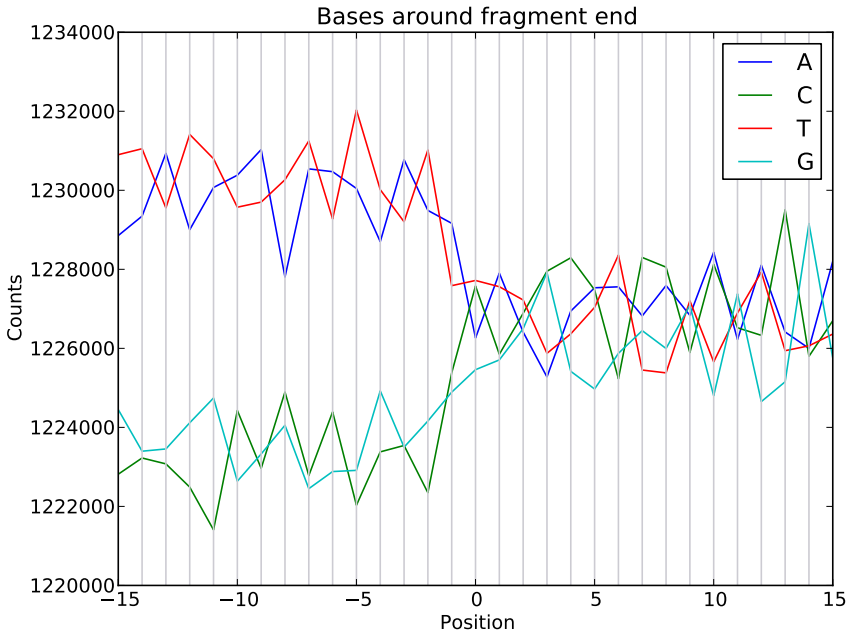
$$\epsilon = \epsilon_{g_i} \times \epsilon_{l_i} \tag{5}$$

For example, the GC and length components above lead to the following joint efficiency mapping:

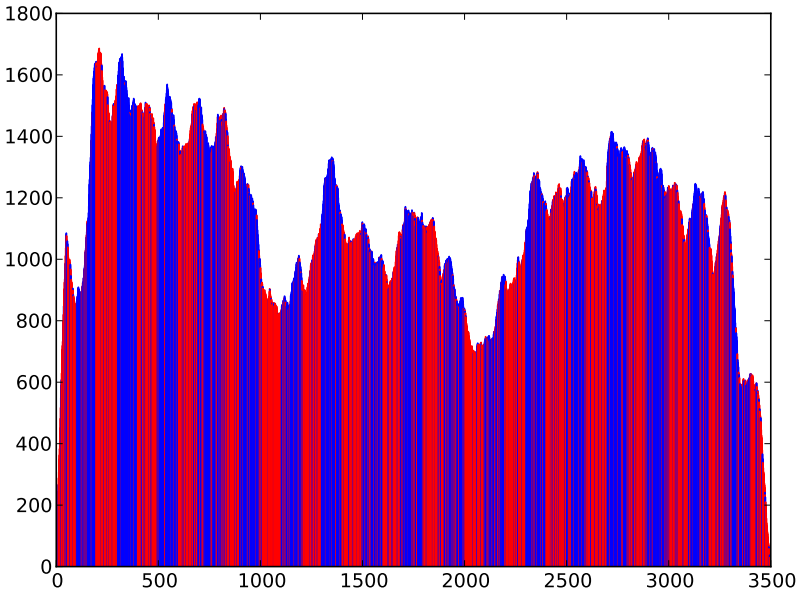


The simulated PCR amplification will create a preference towards fragments with certain GC contents. For example the first simulation setting uses a GC efficiency function preferring fragments with low GC content, which manifests in the output as a sequence specific bias over the whole length of the fragment:



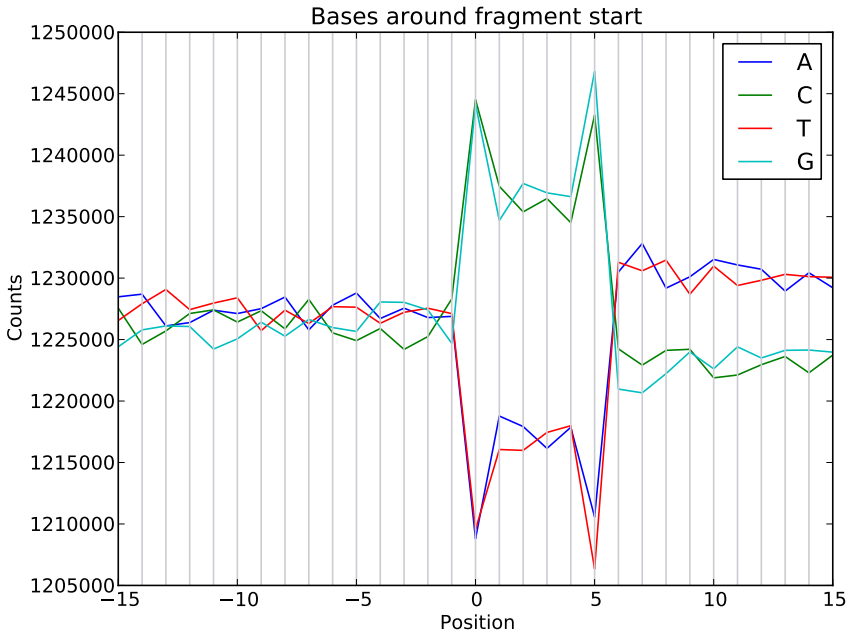


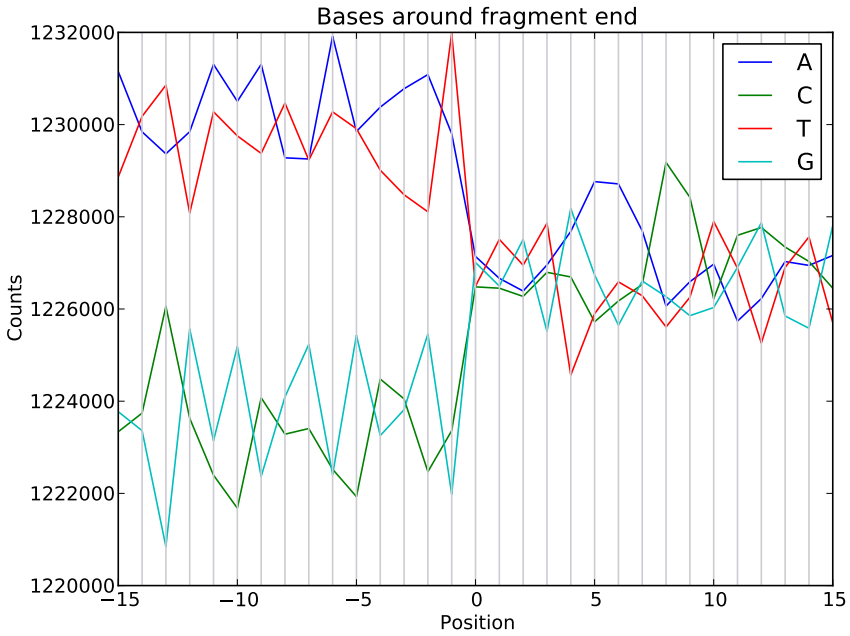
The preference for regions with low GC content can be spotted in the base coverage trends in the output of the second simulation setting as well. Note how simulated PCR alone can create complex trends in base coverage:



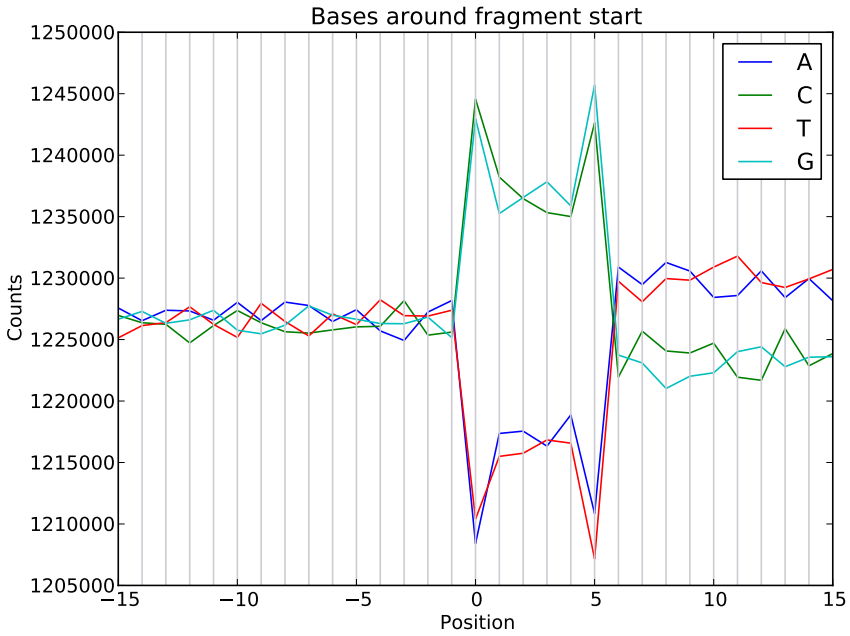
2.2.8 Interactions of simulated biases

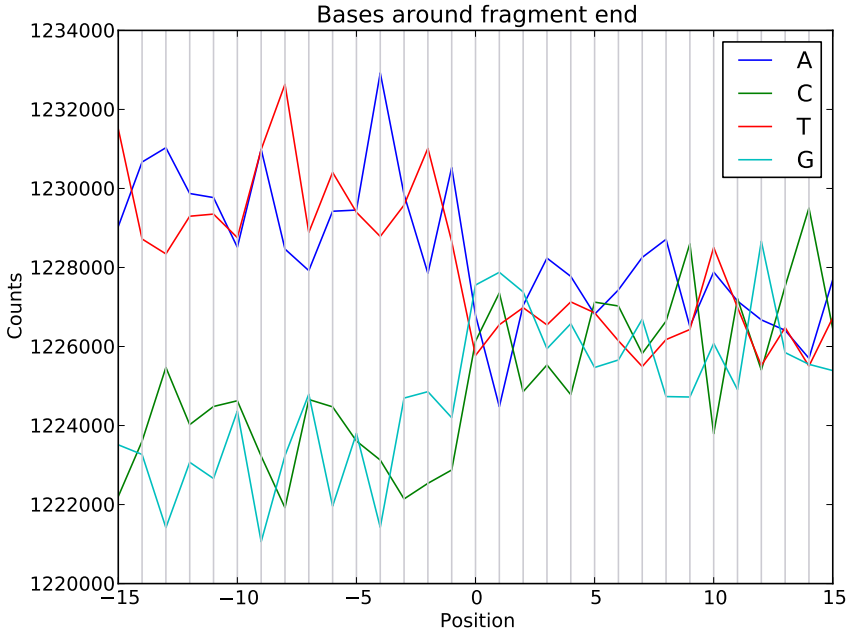
Biases due to the fragmentation process, priming and PCR simulation can interact in a complex way. In most cases the output of such simulations will require interpretation by statistical approaches. However, in the idealised case of the first simulation setting we can enable both priming and PCR simulation and observe how these factors combine in order to create a more complex sequence specific bias:





In the simulation settings above the simulation of polyadenylation was practically disabled by setting the tail length to 1. Now if besides the priming and PCR simulation we also simulate longer tails with a mean length of 200, we can also observe how this leads to a slight increase in the frequency of A nucleotides after the sixth base of the fragment:





This increase in A frequencies is likely to due to the spurious mappings of the A-rich reads coming from the poly(A) tails. This effect seems to have an impact regardless of the relatively long simulated read length (75 bases).

In the light of the relatively complex bias patterns produced by the simple simulations above it is clear how using more realistic parameters and real transcriptomes can produce convoluted biases which allow for the benchmarking of advanced bias correction methods.

2.3 Running the program

```
Usage:
    rlsim [arguments] [transcriptome fasta files (optional)]
```

2.3.1 Input files

The input files specified after the command line arguments are the transcriptome fasta files with the transcript expression levels encoded in the sequence names using `$` as separator. So a transcript named `trA` with expression level 100 has the following fasta record:

```
>trA$100
GCTTCCGGCATCTGGCTCAGTTCCGCCATGGCCTCCTTGGAAGTCAGTCGTAGTCCTCGCAGGTCTCGGCGGGAGCTG ...
...
```

If no input files are specified, `rlsim` will attempt to read the input transcriptome from the standard input.

2.3.2 Command line arguments

Command line arguments not detailed in the previous sections:

`-n` The number of requested fragments (no default, required if no raw parameter file is specified).

`-b` The probability of sampling the fragment from the forward strand (default: 0.5).

`-c` The number of simulated PCR cycles (default: 11).

`-flg` The probability of losing a fragment after fragmentation and before PCR simulation (default: 0.0).

`-m` Multiplier to increase the expression level of all transcripts and so decrease the sampling ratio (default: 1.0).

`-r` The name of the report JSON file (default: `rlsim_report.json`).

`-t` The maximum number of cores to use. It is used to set the `runtime.GOMAXPROCS` variable, also limits the number of gorutines spawned (default: 4).

`-g` Keep the fragments in memory instead of using a disk cache. This speeds up the simulation, but it is only usable if the input transcriptome is small or if there is sufficient memory available (default: false).

`-si` Initial random seed (default: set from UTC time).

`-sp` Random seed used for PCR amplification and sampling (default: seeded from the initial RNG).

`-ss` Random seed used for fragment sampling (default: seeded from the PCR RNG).

`-v` Toggle verbose messages (default: false).

`-h` Display help message and exit.

`-V` Display version and exit.

`-prof` Enable CPU profiling and write info to the specified file (default: "").

`-gcfreq` Explicitly trigger garbage collection after this many transcripts (defaults: 100).

`-rng` Generate test output for the random number generators (default: off).

2.3.3 Output

`rlsim` produced the following output:

- The simulated fragments are written out to the standard output in fasta format. The format of the name field is similar to the one used by the `simLibrary` tool:

  $$>\text{Frag\_}\textit{frag\_nr}\ \textit{transcript\_name}\ (\text{Strand}\ \textit{strand}\ \text{Offset}\ \textit{start} - \textit{end})$$

- The `rlsim` report file (named `rlsim_report.json` by default) contains various information about the simulation in JSON format. It is parsed by the `plot_rlsim_report` tool, transforming it into a PDF report.

If invoked with the `-v` flag, `rlsim` outputs verbose messages about the ongoing simulations. Most of the messages are self-explanatory, but some of them are worth mentioning in detail:

- The "**total number of fragments in the pool**" is the number of fragments in the fragment pool after simulating PCR.
- The "**sampling ratio**" is the number of sampled fragments divided by the total number of fragments in the pool. This is important, as in many cases the interactions of the different biases can be modulated by this ratio. The sampling ratio under a fixed input transcriptome can be modified by setting the fragment loss probability through the `-flg` flag. Higher fragment loss probabilities lead to smaller fragment pools and so higher sampling ratios.
- In some cases the input transcriptome and the specified parameters produce a fragment pool which does not have enough fragments in some length categories requested by the user via the total number of fragments and the target size distribution. These are reported by `rlsim` as "**missing fragments**".

2.3.4 Advanced examples

Simulating PCR pseudo-replicates

`rlsim` allows the user to change the random seeds used at different points during simulation, which allows for the simulation of pseudo-replicates of different kinds. For example, PCR pseudo-replicates are simulated in the following steps:

1. Run a simulation with the initial and PCR random seeds specified via the `-si` and `-sp` flags:

   ```
   $ ./rlsim -n 2000 -si 112 -sp 200 test/basic/test_transcripts.fas > frags_rep1.fas
   ```

2. Run a second simulation with the same input and parameters and the same initial, but a *different PCR random seed*:

   ```
   $ ./rlsim -n 2000 -si 112 -sp 300 test/basic/test_transcripts.fas > frags_rep2.fas
   ```

Using the same input and initial random seed ensures that the composition of the fragment pools after fragmentation is exactly same in the two simulations. The random seed specified through the `-sp` flag will initialise a new random number generator used throughout PCR simulation and fragment sampling. By specifying different seeds we can simulate distinct PCR amplifications of the same fragment pool and the subsequent sampling of fragments. This allows for assessing the effects of variation introduced by the respective processes.

WARNING: Note, that for the different PCR pseudo-replicates the input fragment pool is exactly the same. In real experiments this pool is split into sub-pools which serve as templates for the PCR replicates. For practical reasons, `rlsim` does not simulate this splitting process (for that we call the output "pseudo-replicates"), however this is probably is safe to ignore in most of the cases.

Simulating sampling pseudo-replicates

A sampling pseudo-replicate is equivalent to technical replicates performed by sequencing the output of a given library many times. Again, the `rlsim` replicates are not proper replicates, as in contrary to real experiments they are performed *with replacement*. However, if the sampling ratio is sufficiently small, again, this can be safely ignored.

The sampling pseudo-replicates are simulated as following:

1. Run a simulation with the initial and sampling random seeds specified via the `-si` and `-sp` flags:

   ```
   $ ./rlsim -n 2000 -si 212 -ss 400 test/basic/test_transcripts.fas > frags_rep1.fas
   ```

2. Run a second simulation with the same input and parameters and the same initial, but a *different sampling random seed*:

   ```
   $ ./rlsim -n 2000 -si 212 -ss 600 test/basic/test_transcripts.fas > frags_rep2.fas
   ```

The output of the two simulations are fragments sampled from the same fragment pool.

3 Estimating parameters using `effest`

3.1 Quick reference

```
usage: effest [-h] [-f ref_fasta] [-i iso_list] [-c nr_cycles] [-m mean_eff]
              [-M max_eff] [-d dist_fam] [-g out_fasta] [-j out_json]
              [-e expr_mul] [-a] [-t] [-w step_size] [-s out_count_file]
              [-k in_count_file] [-p out_prior_file] [-o in_prior_file]
              [-q min_qual] [-r report_file] [-l log_file] [-v]
              [input file]

Estimate GC-dependent fragment amplification efficiencies and fragment size
distribution from paired-end RNA-seq data mapped to transcriptome (version
1.1).

positional arguments:
  input file         Aligned *paired end* reads in SAM format sorted by
                     *name*.

optional arguments:
  -h, --help         show this help message and exit
  -f ref_fasta       Reference fasta.
  -i iso_list        List of single isoform transcripts.
  -c nr_cycles       Number of PCR cycles (11).
  -m mean_eff        Assumed pool efficiency (0.87).
  -M max_eff         Assumed maximum efficiency (None).
  -d dist_fam        Distribution to model fragment size distribution
                     (n|sn|*auto*).
  -g out_fasta       Output fasta.
  -j out_json        File to store estimated raw parameters (raw_params.json).
  -e expr_mul        Expression level multiplier (10000.0).
  -a                 Do not use GC efficiency correction on expression levels
                     (False).
  -t                 Trim off old expression values (True).
  -w step_size       Sliding window size / step size ratio (5).
```

```
-s out_count_file  Pickle counts to the specified file (effest_counts.pk).
-k in_count_file   Load counts from specifies file.
-p out_prior_file  Pickle fragment prior to the specified file
                   (effest_pr.pk).
-o in_prior_file   Load fragment prior from the specified pickle file.
-q min_qual        Minimum mapping quality (0).
-r report_file     Report PDF (effest_report.pdf).
-l log_file        Log file.
-v                 Toggle verbose mode (False).
```

3.2 Background

3.2.1 Scope and limitations

The implicit simulation model implemented in `rlsim` was designed in order to be more complex than the models used for inference in order to provide the means for fair benchmarking, hence the joint inference of all parameters would not be practical under this model. However, the most likely use case of an RNA-seq library simulator is to simulate datasets which reproduce the properties of real datasets. For that reason, the `rlsim` package provides `effest` , a tool to estimate *some* parameters using `ad-hoc` procedures.

`effest` provides an estimate of the empirical fragment size distribution, GC-dependent amplification efficiencies based on a default or user-provided mean amplification efficiency, *relative* expression levels. It does not provide any estimate of length-dependent amplification efficiencies as inferring those would be hard from data generated by standard RNA-seq experiments.

WARNING: The estimates provided by `effest` are based on naive assumptions as they ignore the interactions between the different experimental factors. However, they are good starting parameters when trying to replicate the properties of a given dataset.

The `effest` tool is implemented in `Python`, relying on the `HTSeq`, `NumPy` and related packages.

3.2.2 Estimating the fragment size distribution

The fragment size distribution is analysed in the following steps:

- The empirical fragment size distribution is estimated by simply normalising the fragment counts in each length category.
- The truncated normal and skew normal distributions are fitted to the empirical distribution using a maximum likelihood procedure.
- The best fitting analytical distribution is then selected using the Akaike information criterion (AIC) [14].

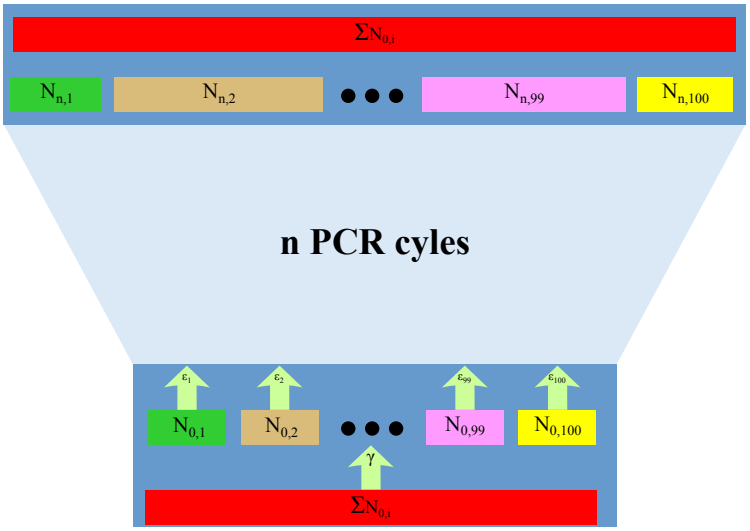
The empirical distribution is reported in the raw parameters JSON file, while the best fitting analytical distribution is reported as a part of the suggested `rlsim` command line.

3.2.3 Estimating GC-dependent amplification efficiencies

Before discussing the details of how `effest` estimates the GC-dependent efficiencies from a single paired-end RNA-seq dataset, we first present the theory behind estimating these parameters in the case of ideal custom experiments. This is necessary in order to illustrate the limitations imposed by a single standard dataset and to understand the motivation behind the procedures used by the tool in order to circumvent these limitations.

Idealised PCR experiment with measurable counts

The figure below illustrates a PCR experiment where we can measure the fragment counts exactly:



The fragments are binned in 100 GC-content bins, $N_{i,j}$ being the number of fragments in the j^{th} GC content bin during the i^{th} cycle (e.g. $N_{0,50}$ is the number of fragments with 50% GC content before PCR amplification). Each GC content bin has an associated amplification efficiency ϵ_i.

PCR amplification on the level of individual bins is modelled as a Galton-Watson [15] branching process with constant efficiencies across cycles, so the expected number of fragments after n cycles in a given bin can be calculated as:

$$E[N_{n,i}] = N_{0,i}(1+\epsilon_i)^n \tag{6}$$

In an idealised PCR experiment where we can measure the fragment *counts* in individual bins after cycles n_1 and n_2 the estimation of the amplification efficiencies (ϵ_i) would be trivial. However, RNA-seq experiments allow only for the measurement of *proportions*, hence it is necessary to consider the total number of fragments after n cycles.

Serial measurements of fragment proportions and known pool efficiency

In order to account for the increase in the size of the fragment pool, we define a Galton-Watson process which acts on all of the fragments (the "fragment pool") with a "pool efficiency" γ satisfying the following equation:

$$\left(\sum N_{0,i}\right)(1+\gamma)^n = \sum\left[N_{0,1}(1+\epsilon_i)^n\right] \tag{7}$$

Note that despite the fact that γ cannot be expressed as a simple linear combination of the individual efficiencies, it is always possible to find a value satisfying the equation above.

For now, assume that γ is know, necessarily obtained from a measurement independent from the fragment proportions calculated from RNA-seq data. Then the number of fragments in the whole pool after n cycles can be calculated as:

$$E[\sum N_{n,i}] = \left(\sum N_{0,i}\right)(1+\gamma)^n \tag{8}$$

Following equations 6 and 8 the *proportion* of the fragments from the j^{th} bin relative to the whole fragment pool after n PCR cycles can be calculated as:

$$\pi_{n,j} = \frac{N_{0,j}(1+\epsilon_j)^n}{(\sum_k N_{0,k})(1+\gamma)^n} \tag{9}$$

Then the ratio of proportions after n_2 and n_1 cycles is:

$$\frac{\pi_{n_2,j}}{\pi_{n_1,j}} = \frac{(1+\epsilon_j)^{n_2-n_1}}{(1+\gamma)^{n_2-n_1}} \tag{10}$$

By rearranging the equation above, we get an expression which can be used for estimating amplification efficiencies:

$$\hat{\epsilon}_j = \exp\left[\frac{1}{n_2 - n_1} \ln\left(\frac{\pi_{n_2,j}}{\pi_{n_1,j}}\right)\right](1+\gamma) - 1 \tag{11}$$

For some bins the ratio of proportions will be necessarily less then one, in which case we set the amplification efficiency to 0 or some arbitrary value.

The pool amplification efficiency (γ)

In standard RNA-seq datasets there is no information about the pool amplification efficiency (γ), so this has to be specified by the user. *In principle*, it should be possible to measure the "average" amplification efficiency of the fragment pool using quantitative PCR methods. The pool efficiency can be directly set via the `-m` flag, alternatively the user can specify the assumed maximum amplification efficiency through the `-M` flag.

The default is a pool efficiency (`-m`) of 0.87, in line with the amplification efficiencies reported in the quantitative PCR literature for individual amplicons [29, 24, 27]

Inferring efficiencies from a single dataset

Standard RNA-seq protocols usually sample the fragments from the amplified fragment pool only after a specified PCR cycle n_1, hence there is only a single set of fragment proportion ($\pi_{n_1,j}$) available. At the first glance this should make it impossible to estimate GC-dependent efficiencies using the formulas above. In order to circumvent this problem, `effest` employs an *ad-hoc* procedure based on the assumption that the coverage biases are local and hence in the case of long transcripts they "average out" without affecting too much the estimates of per-transcript relative expression levels.

If the assumption above hold, we can calculate a "**fragment prior**", reflecting the proportions in the GC content bins if there were no biases present ($\pi_{0,j}$) using the following procedure:

- Estimate the fragment size distribution as described above.
- For every length category, do a sliding window analysis over a list of specified transcripts. Tabulate the GC content weighted by the relative expression level of the respective transcript and the probability of a fragment having the respective length. Step size used during the sliding window analysis is calculated as the ratio of the fragment size and the value of the `-w` flag (10 by default).
- From the counts resulting from the sliding window procedure, calculate the expected marginal GC content distribution under the assumption of no PCR biases, which is used as proxy for the proportions in the GC content bins before amplification ($\pi_{0,j}$).

The amplification efficiencies then are calculated using equation 11 based on the proportions calculated from the data ($\pi_{n_1,j}$), the fragment prior ($\pi_{0,j}$) and the pool efficiency specified by the user (γ).

WARNING: Note that the approach for estimating GC-dependent amplification efficiencies above is rather naive as it ignores the interactions of amplification biases with priming biases and seize selection and it is based on a strong assumption of locality of coverage biases. However, simulations suggest that if the assumption hold, the approach performs well in recovering efficiencies.

Fitting the GC efficiency function

After the GC-dependent amplification efficiencies have been estimated as described above, `effest` attempts to fit the GC efficiency function used in `rlsim` (equation 3). The maximum efficiency is set directly as the observed maximum value, while the minimum efficiency and the shape parameter is obtained by a least squares procedure using the weights $\epsilon_j(1-\epsilon_j)$. The weight reflects the fact that usually most of the fragments in real experiments do not have extreme GC contents, hence we expect more reliable data around 50% of GC content.

The estimated parameters are reported as part of the suggested `rlsim` command line parameters.

3.2.4 Estimating relative expression levels

The counts of the fragments mapped to a given transcript t are binned by GC content and stored after parsing in a count vector C_t. Then, based on the estimated GC-dependent efficiencies, an "efficiency weight" vector (w) is calculated for each GC content bin, based on the expected number of descendants of a single molecule from the bin after n PCR cycles:

$$w_i = \frac{1}{(1+\epsilon_i)^n} \tag{12}$$

The relative expression level of the transcript (E_t) is calculated by summing the count vector normalised by the length of the transcript (l_t) and the efficiency weight vector and also multiplied by a value supplied by the user through the `-e` flag (E_u):

$$E_t = \left(\sum_{i=0}^{100} C_{t,i} \frac{w_i}{\min(w)} \right) \frac{1}{l_t} E_u \tag{13}$$

Note that the GC-dependent efficiency correction through w_i gives more weight to fragments with lower efficiencies.

The calculated relative expression level is analogous to the widely used FPKM metric, but corrected for the effects of GC-dependent efficiencies.

If the `-a` flag is specified, `effest` will also save uncorrected ("flat") expression levels in a separate fasta file with the `flat_` prefix. The uncorrected expression levels are multiplied with a constant (κ) in order to make their overall magnitude similar to the corrected levels:

$$E_t^f = \left(\sum_{i=0}^{100} C_{t,i} \kappa \right) \frac{1}{l_t} E_u \tag{14}$$

As it is impossible to estimate absolute expression levels from standard RNA-seq data, the estimates provided by the above approach can be too high or too small for a particular simulation setting. Hence, the purpose of the E_u multiplier specified through the `-e` flag (10^5 by default) is to globally adjust the expression levels. Using the pickle files generated by `effest` as input (see section 3.4) makes it feasible to tune the relative expression levels, by skipping the parsing of the input data and the generation of fragment prior, which are the most time-consuming steps.

WARNING: Always examine carefully the distribution of relative expression levels and expression levels of individual transcripts in order to make sure that they are suitable for you particular simulation setting!

3.3 Running the program

```
usage: effest [-h] [-f ref_fasta] [-i iso_list] [-c nr_cycles] [-m mean_eff]
              [-M max_eff] [-d dist_fam] [-g out_fasta] [-j out_json]
              [-e expr_mul] [-a] [-t] [-w step_size] [-s out_count_file]
              [-k in_count_file] [-p out_prior_file] [-o in_prior_file]
              [-q min_qual] [-r report_file] [-l log_file] [-v]
              [input file]

Estimate GC-dependent fragment amplification efficiencies and fragment size
distribution from paired-end RNA-seq data mapped to transcriptome (version
1.1).

positional arguments:
  input file          Aligned *paired end* reads in SAM format sorted by
                      *name*.
```

3.3.1 Input files

The input data is **name-sorted paired-end** reads in SAM format, mapped to the reference transcriptome specified via the `-f` flag.

The file containing the list of single-isoform transcripts (one name per file) is specified through the `-i` flag. If no file is specified, the full set of transcripts is used for calculation of the fragment size distribution and GC-dependent efficiencies.

3.3.2 Command line arguments

`-h` Show help message and exit.

`-f` Reference transcriptome in fasta format (required, no default)

`-i` The list of single isoform transcripts used for estimating GC-dependent amplification efficiencies and fragment size distribution (no default).

`-c` Number of PCR cycles used in calculations (default: 11).

`-m` The assumed pool efficiency (γ, default: 0.87)

`-M` The efficiency assumed for the bin with the highest increase in proportion. Overrides the `-m` flag when specified.

`-d` Distribution family to be fitted on the empirical distribution of fragment sizes. The possible values are `n` for truncated normal, `sn` for skew normal and `auto` for selecting the best fitting family using AIC (default).

`-g` The name of the output fasta file, augmented with relative expression levels (default: <input_name>_expr.fas).

`-j` JSON file to store estimated raw parameters (default: `raw_params.json`).

`-e` The value of the expression level multiplier (E_u, default: 10^4)

`-a` Disable GC-dependent efficiency correction (False).

`-t` Do not trim off old expression values from input transcriptome.

`-w` Sliding window step size parameter. Step size used for the sliding window analysis is equal to the fragment size divided by the value of this flag (10 by default). Set it to smaller value in order to speed up the calculating of the fragment prior.

`-s` The name of the fragment count pickle file (default: `effest_counts.pk`).

`-k` Use the fragment counts pickled in the specified file as input (no default).

`-p` Pickle fragment prior to the specifies file (default: `effest_pr.pk`).

`-o` Load fragment prior from the specified pickle file (no default).

`-q` Minimum mapping quality. Fragment having either reads a mapping quality lower than this value are discarded (default: 0)

`-r` Name of the report PDF file (default: `effest_report.pdf`).

`-l` Name of the log file (default: stderr).

`-v` Toggle verbose mode.

3.3.3 Output files

- The suggested rlsim parameters are reported to the standard output:

```
Suggested rlsim parameters:
-d "1.0:sn:(364, 132, 2.80662, 190, 925)"
-eg "(4.95603, 0.518537, 1)"
-n 1880365
```

- The relative expression levels are reported in the fasta file `<input_name>_expr.fas`, the estimated "raw" parameters are saved in a JSON file.
- The report file contains the following plots:
  - the marginal GC content distribution
  - the marginal fragment size distribution
  - the joint GC content/fragment size distribution
  - the "fragment prior"
  - the "marginal GC content prior"
  - a plot of the estimated GC-dependent efficiencies along with the fitted efficiency function
  - the distribution of estimated relative expression levels (also with the zero value excluded)
  - the plots of the truncated normal and skew normal distributions fitted to the fragment size distribution

3.4 Re-using pickles of parsed data

Instead of parsing again the SAM file (which is quite time consuming), `effest` allows for using the counts pickle file generated during parsing. Specifying the matching reference transcript file is still necessary:

```
./effest -v -f test/ref.fas -k test/effest_counts.pk
```

Using also a pickled fragment prior generated by a previous run makes the analysis fast, which facilitates reanalysis with different parameters:

```
./effest -v -f test/ref.fas -k test/effest_counts.pk -o effest_pr.pk
```

4 Additional tools

4.1 `plot_rlsim_report`: plotting `rlsim` reports

```
usage: plot_rlsim_report [-h] [input file]

Plot rlsim report (version 1.0).

positional arguments:
  input file  rlsim report file.

optional arguments:
  -h, --help  show this help message and exit
```

This tool takes the `rlsim` report file and generates a PDF report of the simulation details.

4.2 sel: sampling expression levels

```
usage: sel [-h] [-t] [-d dist_param] [-b nr_bins] [-r report_fil]
           [input fasta file]

Sample expression levels from a mixture of gamma distributions (version 1.0).

positional arguments:
  input fasta file  Transcripts and expression levels in Fasta format.

optional arguments:
  -h, --help        show this help message and exit
  -t                Trim sequence names.
  -d dist_param     Expression level distribution.
  -b nr_bins        Number of bins in histogram.
  -r report_fil     Report PDF file.
```

This tool takes a fasta file as input and produces fasta files augmented with the expression levels sampled from a mixture of gamma distributions specified by the -d flag (default: `"0.5:g:(5e3,0.5) + 0.5:g:(5e4,1000)"`). It also produces a report PDF with a histogram of the sampled expression levels.

4.3 pb_plot: plotting sequence biases

```
usage: pb_plot [-h] [-f ref_fasta] [-r report_file] [-w winsize] [-i tr_list]
               [-q min_qual] [-p pickle_file] [-s]
               [input file]

Visualise sequence biases around fragment start/end (version 1.1).

positional arguments:
  input file      Aligned *paired end* reads in SAM format.

optional arguments:
  -h, --help      show this help message and exit
  -f ref_fasta    Reference sequences in fasta format.
  -r report_file  Name of PDF report file.
  -w winsize      Window size.
  -i tr_list      List of single isoform transcripts.
  -q min_qual     Minimum mapping quality.
  -p pickle_file  Results pickle file.
  -s              Assume single ended dataset.
```

This tool plots the counts of the bases (controlled by the `-w` flag) around the start and end of fragments defined by the paired-end reads in the input SAM file. It also reports the GC content distribution of the fragments. The input file must be sorted by read name! See section 2.2.6 for example plots.

4.4 cov_cmp: comparing coverage trends

```
usage: cov_cmp [-h] -f ref_fasta [-g] [-t nr_top] [-c min_cov] [-i iso_list]
               [-l min_length] [-x] [-y] [-r report_file] [-q min_qual]
               [-p pickle_file] [-v] [-s]
               input file input file

Compare relative coverage trends between the *expressed* transcripts of two
datasets (version 1.1).

positional arguments:
```

```
  input file      Two sets of aligned *paired end* reads in SAM format.

optional arguments:
  -h, --help      show this help message and exit
  -f ref_fasta    Reference sequences in fasta format.
  -g              Do not color by AT/GC.
  -t nr_top       Plot at least this many top matches (30).
  -c min_cov      Minimum number of fragments per transcript (20).
  -i iso_list     List of single isoform genes.
  -l min_length   Minimum transcript length.
  -x              Sort by correlation coefficients.
  -y              Plot pairwise cumulative coverage.
  -r report_file  Name of PDF report file.
  -q min_qual     Minimum mapping quality (0).
  -p pickle_file  Results pickle file.
  -v              Toggle verbose mode.
  -s              Assume single ended dataset.
```

The `cov_cmp` tool takes two datasests mapped on the same transcriptome and reports the rank correlation of transcript level fragment counts and the per-transcript rank correlation of the *fragment* count vectors:

```
./cov_cmp -f test/ref.fas test/aln1.sam test/aln2.sam
Per-transcript fragment count rank correlation: rho=0.93007, p=1.17022e-05
Transcript              p-value         rho
tr15$48765              2.96119e-18             0.101709
tr8$13360               2.70926e-33             0.275761
tr6$49718               8.59522e-22             0.101769
tr13$313                0.027806                0.0281846
tr12$50515              2.95321e-38             0.211815
tr9$8395                0.000619112             0.03841
tr10$52401              3.32435e-43             0.171195
tr14$48371              6.27646e-15             0.0910417
tr5$48415               5.6484e-34              0.14148
tr11$47470              2.78874e-156            0.6647
tr7$51463               8.49106e-53             0.377702
```

The PDF report contains various plots aiding the comparison of coverage trends.

4.5 `plot_cov`: plot read coverage colored by reference base

```
usage: plot_cov [-h] -r ref_fasta -b bam [-o outfile]

Plot read coverage colored by the reference base (AT - blue, GC - red). This
tools requires samtools to be installed in path.

optional arguments:
  -h, --help    show this help message and exit
  -r ref_fasta  Reference transcriptome.
  -b bam        Position sorted and indexed BAM file.
  -o outfile    Output PDF (plot_cov.pdf).
```

See section 2.2.6 for example plots.

5 Acknowledgements

The authors would like to give thanks for the useful discussions and suggestions to (in no particular order): Kevin Gori, Christophe Dessimoz, Nick Williams, John Marioni, Nuno Fonseca, Pär Engström,

Paul Bertone. The `rlsim` tool was inspired by the early versions of the Flux Simulator software [21].

References

[1] biopython: Freely available tools for computational molecular biology. URL: `http://pypi.python.org/pypi/biopython`.

[2] BWA: Burrows–Wheeler Aligner. URL: `http://bio-bwa.sourceforge.net`.

[3] The Go programming language. URL: `http://golang.org`.

[4] HTSeq: A framework to process and analyze data from high-throughput sequencing (HTS) assays. URL: `http://pypi.python.org/pypi/HTSeq`.

[5] Installing the Go compiler. URL: `http://golang.org/doc/install`.

[6] matplotlib: Python plotting package. URL: `http://pypi.python.org/pypi/matplotlib`.

[7] NumPy: array processing for numbers, strings, records, and objects. URL: `http://pypi.python.org/pypi/numpy`.

[8] PyPI - the Python Package Index. URL: `http://pypi.python.org/pypi`.

[9] The `rlsim` download page. URL: `https://github.com/sbotond/rlsim/downloads`.

[10] The `rlsim` GitHub repository. URL: `https://github.com/sbotond/rlsim`.

[11] The SAMtools homepage. URL: `http://samtools.sourceforge.net`.

[12] SciPy: Scientific Library for Python. URL: `http://pypi.python.org/pypi/scipy`.

[13] simNGS and simLibrary – Software for Simulating Next–Gen Sequencing Data. URL: `http://www.ebi.ac.uk/goldman-srv/simNGS`.

[14] Wikipedia: Akaike information criterion. URL: `http://en.wikipedia.org/wiki/Akaike_information_criterion`.

[15] Wikipedia: Galton-watson process. URL: `http://en.wikipedia.org/wiki/Galton%E2%80%93Watson_process`.

[16] Wikipedia: Gamma distribution. URL: `http://en.wikipedia.org/wiki/Gamma_distribution`.

[17] Wikipedia: Normal distribution. URL: `http://en.wikipedia.org/wiki/Normal_distribution`.

[18] Wikipedia: Skew normal distribution. URL: `http://en.wikipedia.org/wiki/Skew_normal_distribution`.

[19] Yuval Benjamini and Terence P Speed. Summarizing and correcting the GC content bias in high-throughput sequencing. *Nucleic Acids Res*, 40:e72, 2012.

[20] Juliane C Dohm, Claudio Lottaz, Tatiana Borodina, and Heinz Himmelbauer. Substantial biases in ultra-short read data sets from high-throughput DNA sequencing. *Nucleic Acids Res*, 36:e105, 2008.

[21] Thasso Griebel, Benedikt Zacher, Paolo Ribeca, Emanuele Raineri, Vincent Lacroix, Roderic Guigó, and Michael Sammeth. Modelling and simulating generic RNA-Seq experiments with the flux simulator. *Nucleic Acids Res*, 40:10073–83, 2012.

[22] Kasper D Hansen, Steven E Brenner, and Sandrine Dudoit. Biases in Illumina transcriptome sequencing caused by random hexamer priming. *Nucleic Acids Res*, 38:e131, 2010.

[23] Daniel C Jones, Walter L Ruzzo, Xinxia Peng, and Michael G Katze. A new approach to bias correction in rna-seq. *Bioinformatics*, 28:921–8, 2012.

[24] Yann Karlen, Alan McNair, Sebastien Perseguers, Christian Mazza, and Nicolas Mermod. Statistical significance of quantitative PCR. *BMC Bioinformatics*, 8:131, 2007.

[25] Soohyun Lee, Chae Hwa Seo, Byungho Lim, Jin Ok Yang, Jeongsu Oh, Minjin Kim, Sooncheol Lee, Byungwook Lee, Changwon Kang, and Sanghyuk Lee. Accurate quantification of transcriptome from RNA-Seq data by effective length normalization. *Nucleic Acids Res*, 39:e9, 2011.

[26] Jun Li, Hui Jiang, and Wing Hung Wong. Modeling non-uniformity in short-read rates in rna-seq data. *Genome Biol*, 11:R50, 2010.

[27] J M Ruijter, C Ramakers, W M H Hoogaars, Y Karlen, O Bakker, M J B van den Hoff, and A F M Moorman. Amplification efficiency: linking baseline and bias in the analysis of quantitative PCR data. *Nucleic Acids Res*, 37(6):e45, April 2009.

[28] John SantaLucia and Donald Hicks. The thermodynamics of DNA structural motifs. *Annu Rev Biophys Biomol Struct*, 33:415–40, 2004.

[29] G Weiss and A von Haeseler. A coalescent approach to the polymerase chain reaction. *Nucleic Acids Res*, 25(15):3082–7, August 1997.