Nonpareil Documentation

Release 3.5.3

Luis M Rodriguez-R

Jul 08, 2024

CONTENTS

Nonpareil uses the redundancy of the reads in metagenomic datasets to estimate the average coverage and predict the amount of sequences that will be required to achieve "nearly complete coverage".

Contents:

CHAPTER

ONE

INSTALL NONPAREIL

Nonpareil can be installed using conda, as a Biocontainer, with Galaxy, through Homebrew or via the source code.

1.1 Conda installation

1. Install Miniconda
2. Configure the channels to access Bioconda:

```
conda config --add channels defaults
conda config --add channels conda-forge
conda config --add channels bioconda
```

3. Install Nonpareil:

```
conda install nonpareil
```

1.2 Biocontainer

1. Install Docker
2. Pull the container:

```
$ docker pull quay.io/biocontainers/nonpareil
```

3. Launch the container:

```
$ docker run -i -t quay.io/biocontainers/nonpareil /bin/bash
```

1.3 Galaxy

You can install Nonpareil on your own Galaxy instance:

1. Go the Galaxy admin space
2. Search on the main Toolshed for the nonpareil repository available under the "Metagenomics" sections
3. Install it

It will automatically install Nonpareil via the conda installation

1.4 Homebrew

You can install Nonpareil using *Homebrew <https://brew.sh>* or *Linuxbrew <http://linuxbrew.sh/>*.

1. Install *Homebrew <https://brew.sh>* if you haven't yet:

```
$ /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
↪HEAD/install.sh)"
```

2. Get the *brewsci/bio <https://brewsci.github.io/homebrew-bio/>* tap if you haven't yet:

```
$ brew tap brewsci/bio
```

3. Install Nonpareil:

```
$ brew install nonpareil
```

1.5 Source code installation

1.5.1 System requirements

Nonpareil binary: Nonpareil requires a C++ compiler. It has been tested on 64-bit machines with GCC versions >=4.2.1, running Mac OSX and Red Hat Linux.

Nonpareil MPI: If you want to compile Nonpareil with MPI support, you will need OpenMPI (v > 1.4.3 tested). Other implementations of MPI could work, but are yet untested.

Nonpareil utilities: Requires R. No additional libraries are necessary.

1.5.2 Compilation

1. **Get the source**

   Clone the repository from GitHub:

```
git clone git://github.com/lmrodriguezr/nonpareil.git
```

   If you don't have git, you can also download the TAR-Ball and unpack it with:

```
tar zxvf nonpareil.tar.gz
```

2. **Compile**

   Change directory into the newly created folder, and compile Nonpareil:

```
cd nonpareil
make
```

   If you want to compile Nonpareil MPI (see also *MPI support*), just run:

```
make nonpareil-mpi
```

   In either case, you can specify the C++ compiler to be used setting the `cpp` or `mpicpp` variables, respectively. For example:

```
# This compiles nonpareil with /usr/local/bin/g++
make cpp=/usr/local/bin/g++ nonpareil
# This compiles nonpareil-mpi with /usr/local/bin/mpic++
make mpicpp=/usr/local/bin/mpic++ nonpareil-mpi
```

3. **Install**

   If you want to make Nonpareil available system-wide, just run:

   ```
   sudo make install
   ```

   If you don't have superuser privileges and/or want to install Nonpareil in a location other than `/usr/local`, simply set the prefix, for example:

   ```
   make prefix=$HOME/apps install
   ```

   You can also change the location of `R`, if it's not in the `$PATH` or you want to use a non-standard installation:

   ```
   make prefix=$HOME R=~/bin/R install
   ```

   Other variables you can set explicitly for the `install` target are `bindir` (binaries directory) and `mandir` (documentation directory).

CHAPTER

TWO

PREPROCESS THE READS

Nonpareil expects that the sequencing error is always well below 5%, so we suggest using an expected error cutoff of 1% (*i.e.*, Q>20, or 1 error in 100 nucleotides). We recommend to perform this task using SolexaQA.

Ideally, the reads should be in FastA format (althought Nonpareil can read FastQ). To transform FastQ into FastA, you can simply use:

```
# Input: reads.fastq
# Output: reads.fasta
cat reads.fastq | paste - - - - | awk 'BEGIN{FS="\t"}{print ">"substr($1,2)"\n"$2}' >␣
↪reads.fasta
```

Also, if you have paired-end reads, you should use only one sister read per pair in Nonpareil. If you have them interposed in the same file, you can separate them using FastA.split.pl:

```
# Input: reads.fasta
# Output: reads.1.fa and reads.2.fa
FastA.split.pl reads.fasta reads 2
```

CHAPTER

THREE

REDUNDANCY

If you have large files (>1Gb) and access to a cluster, take a look at *MPI support*.

3.1 For the impatient

Even if you're in a hurry, taking a look at *Preprocess the reads* is very important. If you already did, you can simply run:

```
# fastq is recommended for kmer algorithm
nonpareil -s reads.fa -T kmer -f fastq -b output
nonpareil -s reads.fa -T kmer -f fasta -b output

# fasta is recommended for alignment algorithm
nonpareil -s reads.fa -T alignment -f fasta -b output
nonpareil -s reads.fa -T alignment -f fastq -b output

# gzipped files are also supported (with .gz extension)
nonpareil -s reads.fa.gz -T kmer -f fastq -b output
```

Where `reads.fa` is the file containing the trimmed single reads, and `output` is the prefix of the output files to be created.

3.2 Mandatory options

-s <str>	Path to the (input) file containing the sequences. Gzipped files are supported with .gz extension
-T <str>	nonpareil algorithm. can be 'kmer' or 'alignment'.
-f <str>	The format of the sequence: 'fasta' or 'fastq'

3.3 Common options

-b <str> Path to the prefix for all the output files. Replaces the options: -a, -C, -l, and -o; generating files with the suffixes .npa, npc, .npl, and .npo, respectively, unless explicitly set.

-d <num> Subsample iteratively applying this factor to the number of reads, resulting in logarithmic subsampling. Use -d 0 to fall back to linear sampling, controlled by -m, -M, & -i (this was the default before v2.4). By default: 0.7.

-n <int> Number of sub-samples to generate per point. If it is not a multiple of the number of threads (see -t), it is rounded to the next (upper) multiple. By default: 1024.

-L <num> Minimum overlapping percentage of the aligned region on the largest sequence. The similarity (see -S) is evaluated for the aligned region only. By default: 50.

-X <int> Maximum number of reads to use as query. This is capital X. By default, 1,000 reads.

-q <str> Path to the (input) file containing a second dataset to be used as query, for dataset comparisons. This option is currently experimental.

-R <int> Maximum RAM usage in Mib. Ideally this value should be larger than the sequences to analyze (discarding non-sequence elements like headers or quality). This is particularly important when running in multiple cores (see -t). This value is approximated. By default 1024. Maximum value in this version: 4194303

-t <int> Number of threads. Highest efficiency when the number of sub-samples (see -n) is multiple of the number of threads. By default: 2.

-v <int> Verbosity level, for debugging purposes. By default 7. This is lowercase V.

-V Show version information and exit. This is uppercase V.

-h Display this message and exit.

3.4 Additional options

Input/Output

-a <str> Path to the (output) file where all data must be saved. This report is not created by default. See the OUTPUT section.

-C <str> Path to the (output) file where the mating vector is to be saved. This is a capital C.

-F Report the sampled portions as a fraction of the library instead of the number of reads. See -a, -o and the OUTPUT section.

-l <str> Path to the (output) file where the log of the run must be saved. By default the log is sent only to the STDERR. If set, the log is sent to both the STDERR and the log file.

-o <str>	Path to the (output) file where summary is to be saved. By default the summary is sent to stdout (same behavior as using a dash '-'). If an empty string '' is provided, does not produce the summary. See the OUTPUT section.

Sampling

-m <num>	Minimum value of sampling portion. By default: 0.
-M <num>	Maximum value of sampling portion. By default: 1.
-i <num>	Interval between sampling portions. By default: 0.01.

Mating

-c	Do not use reverse-complement. This is useful for single stranded sequences data (like RNA). This is a lowercase C.
-N	Treat Ns as mismatches. By default, Ns (unknown nucleotides) match any nucleotide (even another N).
-S <num>	Similarity threshold to group two reads together. Reducing this option will increase sensitivity while increasing running time. This is uppercase S.
-k <int>	kmer size. You can increase kmer size to increase sensitivity. By default: 24
-x <num>	Probability of taking a sequence into account as query for the construction of the curve. Higher values reduce accuracy but increase speed. This is lower case x. If set, overides -X.

Misc

-A	Autoadjust parameters and re-run. Evaluates the results looking for common problems, adjusts parameters and re-run the analyses. THIS IS EXPERIMENTAL CODE.
-r <int>	Random generator seed. By default current time.

3.5 Input

Sequences must be in FastA or FastQ format. See *Preprocess the reads*.

3.6 Output

Redundancy summary: `.npo` file
: Tab-delimited file with six columns. The first column indicates the sequencing effort (in number of reads), and the remaining columns indicate the summary of the distribution of redundancy (from the replicates, 1,024 by default) at the given sequencing effort. These five columns are: average redundancy, standard deviation, quartile 1, median (quartile 2), and quartile 3.

Redundancy values: `.npa` file
: Tab-delimited file with three columns. Similar to the .npo files, it contains information about the redundancy at each sequencing effort, but it provides ALL the results from the replicates, not only the summary at each point. The first column indicates the sequencing effort (as a fraction of the dataset), the second column indicates the ID of the replicate (a number used only to introduce some controlled noise in plots), and the third column indicates the estimated redundancy value.

Mates distribution: `.npc` file

Raw list with the number of reads in the dataset matching a query read. A set of query reads is randomly drawn by Nonpareil (1,000 by default), and compared against all reads in the dataset. Each line on this file corresponds to a query read (the order is not important). We have seen certain correspondance between these numbers and the distribution of abundances in the community (compared, for example, as rank-abundance plots), but this file is provided only for quality-control purposes and comparisons with other tools.

Log: `.npl` file

A verbose log of internal Nonpareil processing. The number to the left (inside squared brackets) indicate the CPU time (in minutes). This file also provide quality assessment of the Nonpareil run (automated consistency evaluation). Ideally, the last line should read "Everything seems correct". Otherwise, it suggests alternative parameters that may improve the estimation.

CHAPTER

FOUR

NONPAREIL CURVES

The estimation of the *Redundancy* is at the core of Nonpareil, but it's when those values are transformed into average coverage that they become comporable across samples, and become useful for project design and sample evaluation.

To build Nonpareil curves, you need two things. First, the Nonpareil.R file (you can find it in the `utils` folder of Nonpareil). Second, the `.npo` file (or `-o` value, if you used this option) generated in the estimation of *Redundancy*.

4.1 For the impatient

First, load the package. If you don't have it installed yet, you can open R and execute:

```
install.packages('Nonpareil');
library(Nonpareil);
```

If you did *make install* (*Install Nonpareil*), you can simply open R and execute:

```
library(Nonpareil);
```

And you can get help messages using any of:

```
?Nonpareil.curve
?Nonpareil.set
?Nonpareil.legend
?Nonpareil.predict
```

Now, you can simply execute:

```
Nonpareil.curve('output.npo'); # Change output.npo to the actual redundancy file.
```

4.2 Nonpareil.curve()

This function can generate a Nonpareil curve from a `.npo` file. See the documentation of this function inside R after loading the Nonpareil package:

```
?Nonpareil.curve
```

4.3 Nonpareil.set()

This function can generate a plot with several Nonpareil curves from `.npo` files. See the documentation of this function in R after loading the Nonpareil package:

```
?Nonpareil.set
```

Example: I find it very convenient to first prepare a table with the samples, something like:

```
# samples.txt
File        Name    Col
SRS063417.1.L50.npo Posterior fornix        "#FFC8C8"
SRS063287.1.L50.npo Buccal mucosa   "#FF7878"
SRS062540.1.L50.npo Tongue dorsum   "#FF0303"
SRS016335.1.L50.npo Stool   "#C8874C"
SRS015574.1.L50.npo Supragingival plaque    "#E66478"
SRS019087.1.L50.npo Anterior nares  "#DCDC82"
```

Note that this table is tab-delimited, because I find it easier to read, but you can use anything you like (and is supported by R). Next, you can simply type something like this in the R console:

```
library(Nonpareil);
samples <- read.table('samples.txt', sep='\t', header=TRUE, as.is=TRUE);
attach(samples);
nps <- Nonpareil.set(File, col=Col, labels=Name, plot.opts=list(plot.observed=FALSE));
detach(samples);
summary(nps);
```

To execute examples with real data included in the package, you can execute:

```
example(Nonpareil.curve);
example(Nonpareil.set);
```

CHAPTER

FIVE

MPI SUPPORT

Nonpareil supports MPI (Message Passing Interface) since v2.2. This code is stable, but MPI support only covers the alignment kernel, not the k-mer kernel.

5.1 Requirements

You will first need OpenMPI in your computer. There are other MPI implementations, but Nonpareil only supports OpenMPI (by now). Once you have it, you should have at least the C++ compiler (typically `mpic++`) and the interactive executable (typically `mpirun`). If you have the compiler in a non-standard location (for example, to coexist with mpich), change the value of `mpicpp` in the `globals.mk` file. Once you are ready, simply run:

```
cd nonpareil # or wherever you have the nonpareil folder
make nonpareil-mpi
```

That's it. Now you should have the `nonpareil-mpi` binary, that you can place in a location listed in your `$PATH` if you want.

5.2 Running Nonpareil MPI

1. Get your machines ready. If you are familiar with MPI skip directly to #3. If you have your own infrastructure, just make sure they are MPI-capable (network, permissions, software, etc.). If you are using a cluster, just request as many machines as you need (see the resources section below). For example, to request 10 machines with 16 CPUs each in PBS, use `-l nodes=10:ppn=16`.
2. Obtain the machine names. Just prepare a raw text file with the list of machines you want to use. If you are using PBS, you can do this by running:

   ```
   cat $PBS_NODEFILE | awk 'NR%16==0' > hosts.txt # Change the '16' by the number of␣
   ↪CPUs you are using (the value of ppn).
   ```

3. Run Nonpareil MPI. All you need is to call `nonpareil-mpi` with `mpirun`. For example, if you want to use 10 machines, with 16 CPUs each, and the list of machines is in `hosts.txt`, then run:

   ```
   mpirun -np 10 -machinefile hosts.txt nonpareil-mpi -t 16 -s path/to/your/sequences.
   ↪fasta -b output ...
   ```

   Note that the options of `nonpareil-mpi` are the exact same as for `nonpareil`. Just remember that the value of `-t` is the number of threads *per machine*, not the total number of CPUs.

5.3 Resources

If you are interested on MPI, I'm assuming you have big files, so you may be also concerned about resources allocation.

How much memory you will need?

In the Nonpareil 1 paper (Suppl. Fig. 6) you can see the linear relationship between maximum required RAM and the size of the dataset. The function is approximately `RAM = Size + 2`, where `RAM` and `Size` are both in Gb. You can use less RAM than that, and Nonpareil will adapt, but it'll take longer running times. This value is the "maximum required", which means that if you assign more RAM than that, it won't make any difference. Now, that value is the total RAM required. That means that if you use the MPI implementation, you can divide `Size` by the number of computers you are using, and then apply the function above. For example, if you have a 50Gb dataset, you will need (maximum) 52Gb (50 + 2) of RAM for the standard implementation of Nonpareil. However, if you use the MPI version with, for example, 10 machines, you'll need (maximum) 7Gb (50/10 + 2) on each machine.

How many machines you will need?

I don't have a large benchmarking yet for the MPI version, but at the end it really depends on your resources. If you have more machines, it will run faster (unless you have a very small dataset) and it will require less memory (as discussed above).

Should I use more machines or more threads?

Again, it depends on your resources. Multi-threading is (in general) more efficient, because it doesn't have the overhead of network communication. That means that you should favor more CPUs over more machines. However, there are some aspects to take into account. One, as discussed above, is the RAM. More machines = less RAM per machine, while more threads have little impact on RAM usage (actually, more threads = slighly more RAM). Another catch is the resources availability. It is possible that you have tens of machines for your exclusive use, but most likely you are actually sharing resources through a cluster architecture. If you ask for 64 processors per node (assuming you have 64-core machines) you will probably have to wait in queue for quite some time. If you ask for 4 machines, and 64 processors per node, you will likely be waiting in queue for hours or days. However, the same number of threads (256) can be gathered by asking for 16 machines, and 16 processors per node. If you do that, you will give the scheduler more flexibility (note that the nodes=4 ppn=64 is a special case of nodes=16 and ppn=16) hence reducing your queue time. You may be asking: can I simply ask for nodes=256 and ppn=1? Well… you can, but as I said multi-threading is more efficient than multi-nodes, so don't go to the extremes. Also, Nonpareil has three expensive steps:

1. Reading the fasta, which is strictly linear: only one thread is used in only one machine. This process is linear in time with the size of the input file.
2. Comparing reads, which is threaded and multi-node. This is by far the most expensive step, and it is distributed across machines and across CPUs on each machine. This process is linear in time with the size of the input file.
3. Subsampling, which is threaded but not multi-node. This step is not too expensive, and it's nearly constant time. With default parameters, it takes about 2 minutes with 64 threads, but it grows if you reduce `-i`. The time on this step is reduced by more threads (`-t`), but not by more machines.

How can I evaluate the performance in pilot runs?

I must say: I rarely do pilot runs. However, I'm often interested on performance for future runs (for example, for other projects). There are two sources of information that can be handy. One, is the OS itself (or the PBS output file, if you have a good Epiloge configured). For example, to measure the total RAM used, the total walltime, real time, user time, etc. Another source is the .npl file, which contains a log of the Nonpareil run (assuming you used the `-b` option). The number in squared brackets is the CPU time in minutes. Note that the CPU time here is only for the "master" machine. That means: the number of CPU minutes added for all the threads in the main machine. Another useful piece of information is the number of "blocks" used. Ideally, you should have one block per machine; if you have more it means that the RAM assigned (`-R`) was insufficient. You can find it right below the "Designing the blocks scheme…" line. In the ideal scenario (enough RAM), you should have

one Qry block, and as many Sbj blocks as machines (one, if you are not using the MPI implementation). If you have more than that, you could attain shorter running times by increasing the RAM (`-R`).

CHAPTER

SIX

UPDATES HISTORY

6.1 What's new in 3.5

- Gzipped input now supported.
- MPI support extended by using C bindings.
- Safer RAM handling.

6.2 What's new in 3.3

- Release used in the manuscript "Nonpareil 3: Fast Estimation of Metagenomic Coverage and Sequence Diversity".

6.3 What's new in 3.2

- Minor bug fixes and documentation updates.

6.4 What's new in 3.1

- Implemented additional controls for the robustness of the Nonpareil sequence diversity index (Nd).
- Revamped R code, now with Object-oriented structure and uniform R-style options. The new package also includes test data and examples and is now available at *CRAN <https://CRAN.R-project.org/package=Nonpareil>*.

6.5 What's new in 3.0

- New k-mer kernel: A much faster kernel is now provided as an alternative to the alignment-based kernel in previous versions.

6.6 What's new in 2.4

- Subsampling now defaults to logarithmic: Previous versions subsampled linearly, since 2.4 the -d option defaults to 0.7. Options -m, -M, & -i still exist but they are ignored unless -d is 0.
- Nonpareil diversity: A logarithmic value of diversity is now reported, indicating the horizontal position of the Nonpareil curves. This value is the estimated mode of the fitted Gamma CDF, and cannot be calculated if model fitting fails.
- Experimental dataset comparisons: The -q option allows the paired comparison of datasets. The expectation is that the difference between the diversity of a sample by itself (without using -q) and the diversity of the same sample queried with a second sample (with -q) represents the distance between them. Note that this method experimental and is not symetric (hence not a real distance). Usually, the average of the distances in both ways can be used for clustering of samples.

6.7 What's new in 2.3

- Updates history.
- Nonpareil R package: No changes in the functions were introduced, but they are now available as an R package complete with documentation. See *Nonpareil curves*.
- Make's `install` target: This includes the installation of binaries, the manual page, and the new Nonpareil R package.
- Simplified help: Only mandatory and commonly used arguments are now displayed with `nonpareil -h`. Complete documentation is maintained both in the manual page (`man nonpareil`) and the online documentation. In order to maintain a centralized documentation, the complete help messages for the Nonpareil R package are now self-contained, and were removed from the online documentation.