ion torrent

Ion Reporter

Metagenomics 16S algorithms overview

ThermoFisher SCIENTIFIC

Contents

Section 1

Files in the project

Several user-defined settings as well as other parameters are passed into the 16S metagenomics application 3333 by the means of JSON files.

Parameter files

In.json
Contains the actual sample name and the corresponding file name(s). Multiple samples are supported, as well as multiple files per sample.

AppMeta.json
Contains information about the metagenomics application itself (as run in IR). The application only uses the number of CPU cores from this file.

Out.json
This file is produced by the application itself after a completed run, and contains the name of the result HTML file as the “linkout”.

Params.json
Contains the user-selected settings.

Section 2

Java files

The entire application is coded in Java™ 1.7 file as a standard command-line tool. The application will read all input JSON files and start analyzing the supplied BAM input files. The following are the code files used to create the application:

AppIn.java
Reads the input BAM file information from the input JSON file.

AppMeta.java
Reads the metadata from IR.

AppOut.java
Produces the output JSON file.

AppParams.java
Reads the submitted user settings.

BlastRunner.java
Does the actual command-line BLAST™ runs of the supplied sequences.

MergingFunctions.java
Contains all the main functionality of the application:

- Checks the reads in the BAM file
- Checks the reads for primers (if user option is selected)
- Creates a hash table of the unique reads, with copy number information for each one
- Starts the BLAST analysis for all valid reads
- Creates a hierarchy of the finished analyzed reads
- Generates a report for the complete result for all samples and BAM files

Metagenmain.java
Contains the scaffolding for the entire application. Iterates over samples and generates output.

Reports.java
Holds the information generated from reports, and creates the output files needed: HTML, Krona pie chart, FASTA, and TXT files.

Result.java
Small class for holding BLAST result information for FASTA file output of an unmapped sequence.

ResultEntry.java
The node that holds sequence information, as well as links to other connected nodes. In total, this will build the whole hierarchy of nodes—all taxonomy levels needed to produce the result and output.

ResultEntryUniqueRead.java
Holds a unique result after blasting a sequence. Holds the sequence, copy number information, BLAST analysis result (all taxonomy levels), and also computes for the reference.

UniqueSeq.java
Holds the information about a unique sequence before adding it to the hierarchy.

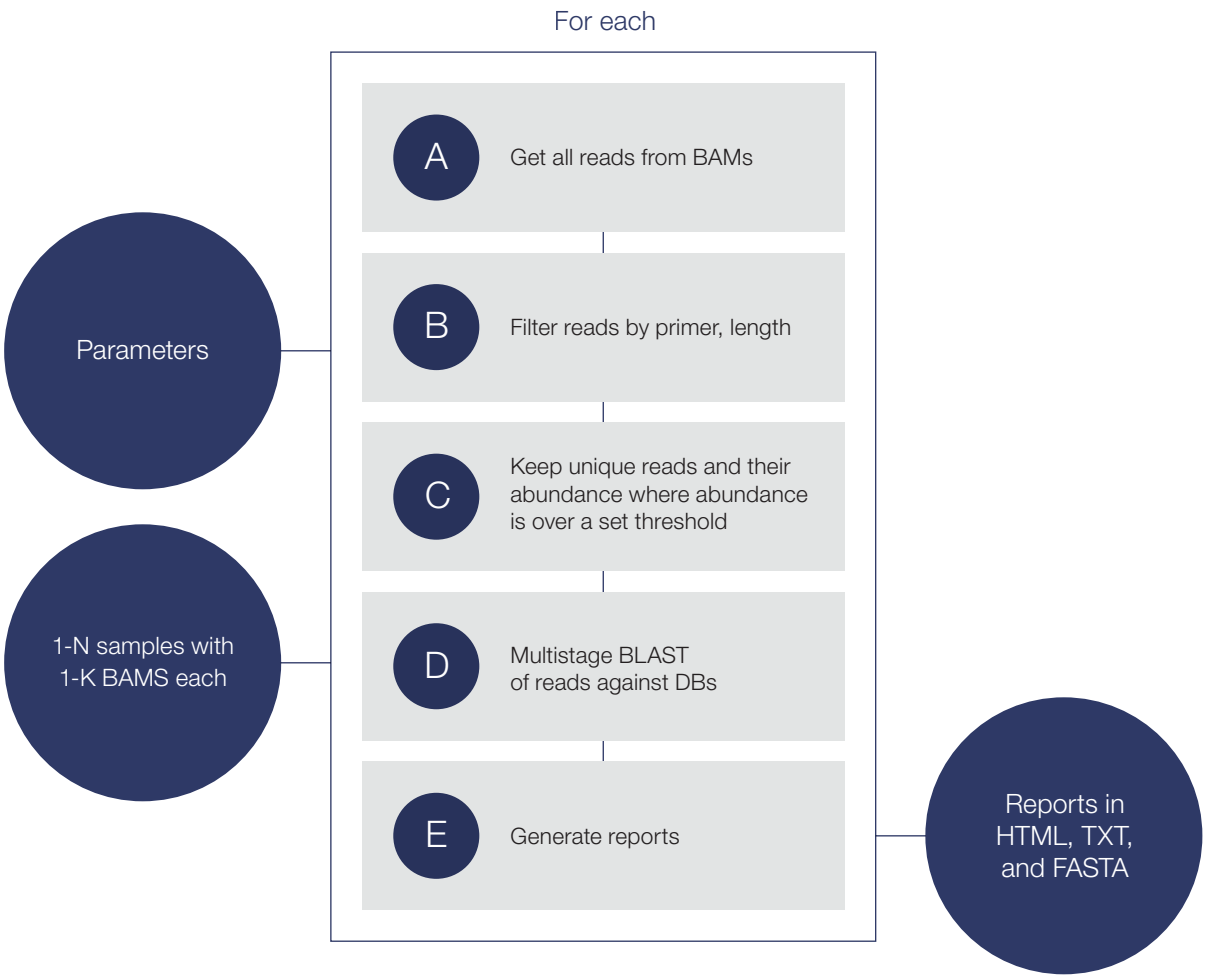
Utils.java
Code for primer trimming, converting and inverting, as well as ambiguous base conversion of a sequence.

ZipUtil.java
Code for creating a ZIP.file (used for reporting files).

Section 3

Algorithms

Outer flow



Inner flow

A. Get all reads from BAMs

Each sample can contain one or more BAM files. All reads from BAMs within a sample will be analyzed together as if coming from one bigger BAM file.

B. Filter reads by primer, length

Using the user set parameters, each read will be trimmed by primer and length. The user can specify whether to require both primers to be present or just one, or no primers at all. After any primer trimming, the resulting read must conform to a certain length.

When trimming, the primer search will use the 15 bases closest to the sequence itself. When searching, three errors can be tolerated. If both primers are needed, the end primer will be used in the same way, with 15 bases and up to three errors.

After primer trimming, each sequence will be trimmed to the closest 20-bp point. That is, if a sequence is 187 bases long, it will be trimmed to 180 bp. This speeds the clustering later in the process.

C. Keep unique reads

After primer trimming and length check, remaining reads will be placed in a hash table. This hash table will contain all the unique reads and their abundance/copy number. When adding a new read to the hash table, the read will have a copy number of 1 if it does not exist in the table already, and if it does, the copy number will be increased by 1.

After adding all reads, the hash table will contain all the unique reads, each with a copy number. A quick filtering will remove all unique reads with a copy number less than the user-set threshold.

Note: There is no quality checking, consensus generation, or other methods to enhance clustering because this could lead to some valid reads being missed and not being analyzed.

D. Multistage BLAST search of reads against DBS

The BLAST mapping engine was chosen for this product. All databases used must conform to a certain format as described later in this document.

Export all unique reads to FASTA files

Each unique read without an OK analysis result will be stored in input files in FASTA format. Each read has a unique ID to keep track of the analysis results throughout. Each file has 250 reads (this number seems to be the most speed-efficient per BLAST process). These files are then passed to a command-line BLAST process. The results returned from each BLAST process will be stored in result files in TXT format.

BLAST search of the FASTA files

The algorithm generates the input files and starts one BLAST process for each file—up to 8 (or the number of CPU cores in the IR VM) concurrently to increase speed.

The specified BLAST settings are:

- E value = 0.01, meaning that any result where the E value is larger than 0.01 will be omitted; all other settings are default or unchanged
- Task = megablast
- Max target seqs = 100, meaning there will be 100 lines maximum in each result for each input read

Other settings in the BLAST command line are to ensure the correct values are returned. The following is an example of an initial BLAST run:

```
blastn -task megablast -evalue 0.01 -query input01.fasta -db blastDBfile -out result01.txt -max_target_seqs
100 -outfmt 6 evalue pident length qstart qend sstart send qseqid stitle
```

Process the results

When all result files have been generated by the BLAST processes, the algorithm continues, checking all the results and adding information such as taxonomy and percent match.

If a unique read does not get an OK result after the BLAST process, the read is still marked as LOW-SCORING or UNMAPPED (if the read did not get a score at all when mapping against this DB). This read will then be reanalyzed with the next stage database. The default is that the analysis starts with the smaller database and then goes to the larger one for reads that did not get a good result in the first one. The algorithm supports 1-N stages, not only 1-2. If a read gets a better result in stage-1 DB mapping than in the next, the best result is used.

If a read does get an OK result, it is marked and not analyzed again in the next stage. After all reads have been checked against stage-1 DB, the UNMAPPED and LOW-SCORING reads are checked against stage-2 DB and so on. After the last stage, any read that did not map at all is exported to an unmapped reads FASTA file. Reads that did map will be added to the result hierarchy. Two more export files are generated: one export file with all the reads that have an OK analysis result (genus- or species-level ID), and a second export file with all the reads that have a non-OK analysis result (family-level ID or worse, including slash call result for genus level).

Active result filters

There are also several "filters" along the way for each result from the BLAST mapping:

- Each result must have good coverage, as set in the user parameters. This means that the resulting mapped sequence must, at least, cover a set percentage of the query sequence.
- If all result lines for one query all have the same species name, down to a set percent range for the percent ID score, the query read is reanalyzed with an increased number of result lines setting. The algorithm starts with 100, then 250, 500, and finally 1,000.
- Reads with a result with an E value greater than 0.01 are not marked as OK.
- If the result for one query read contains more than one species within a set percent range, a slash result name is generated, and the valid result level for the result hierarchy is reduced by one value. For example, if there are two species names within the result, both species names are stored, but the valid level will be 8 instead of 9 (levels are discussed further in the next chapter). The same is repeated if the genus name differs within the set percent range. In this case, where there is more than one genus name, the valid result level would be decreased by two values: one value for the slash species name, and one value for the slash genus name, resulting in a valid level of 7 = family-level ID. In the main result table the user would then just see the family name. If another or different read/sequence has a better differentiation between the two genus/species, this might produce a better result, with only one genus/species in the result. This result (and its copy number) would then be displayed on a separate line in all result tables. The condensed table will not combine the two as they really are different results.
- If the percent match score is higher than a set species threshold (default is 99%), the valid taxonomy level is species-level 9. These results will be shown with a green row background color in the results tables. When looking at the slash calls table, the species slash calls will also be shown with a green color, since the species slash names are in the level-9/species column.
- If the percent match score is lower than a set species threshold, but higher than a genus threshold, the valid taxonomy level is genus level.
- If the percent match score is lower than a set genus threshold, the valid taxonomy level is family level. All unique reads with a family-level result will be reanalyzed in the next DB stage—until the last stage is reached, or a better family-level percent match is obtained.

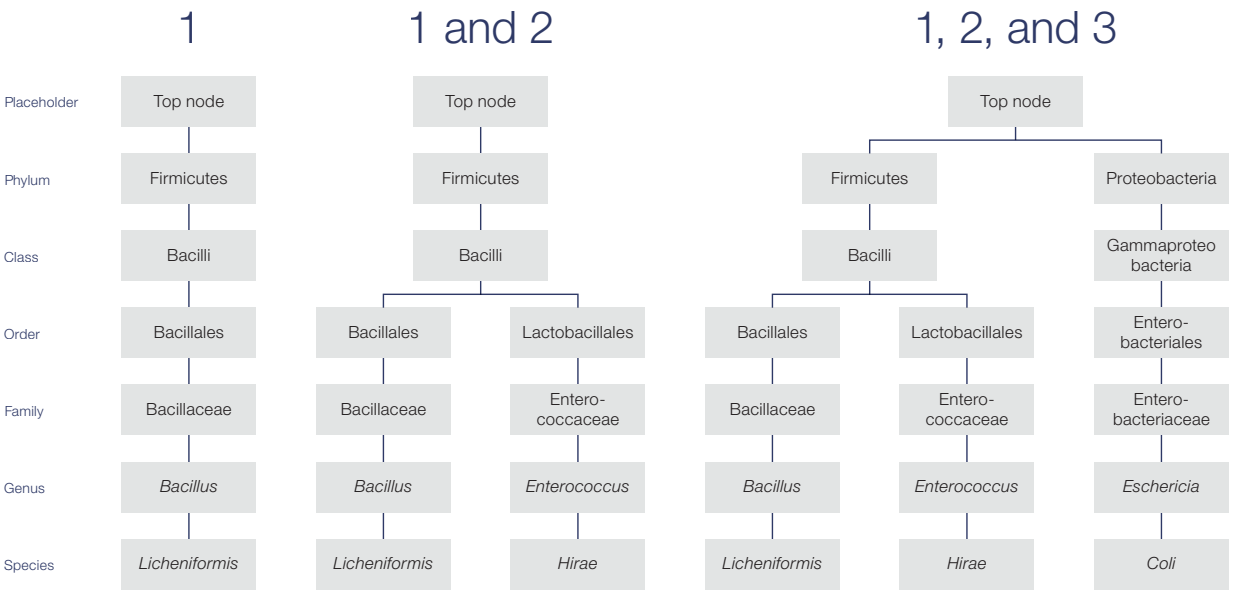
E. Generate reports

In order to create reports where information is listed in various ways, a hierarchy of classes is used. Think of the hierarchy as nodes, where the top node is the top taxonomy result, and the lower nodes are the species results. Each BLAST result has received a percent ID score, as well as a lowest level position. The top-down node tree has 9 levels, where the bottom level is number nine, which is the species level. Each analysis result is just added to the top, and then will trickle down to the correct level, generating new taxonomy nodes on the way if the correct one does not exist already. Each result has a taxonomy entry for each level after BLAST mapping. This taxonomy entry is checked when trickling down the hierarchy.

The nine levels are: primer, domain, kingdom, phylum, class, order, family, genus, and species.
It is possible to specify what levels are to be outputted. In this case we output: phylum, class, order, family, genus, and species—with or without primer at the top.

The following is an example of adding three results to the hierarchy

1. Firmicutes|Bacilli|Bacillales|Bacillaceae|Bacillus|licheniformis|
2. Firmicutes|Bacilli|Lactobacillales|Enterococcaceae|Enterococcus|hirae|
3. Proteobacteria|Gammaproteobacteria|Enterobacteriales|Enterobacteriaceae|Eschericia|coli|



As you see, each added result will generate the needed nodes. The lowest valid level will contain the actual sequence, score, and abundance information.

In order to generate the reports, the recursive algorithm starts at the top node, asking for the abundance information, score, and such. When asked, that node then again asks all nodes directly underneath about the same information. This continues throughout the tree, until all the bottom nodes are reached. All information is then aggregated all the way to the top.

This way the different reports can be generated, outputting information at the requested levels—in various formats.

Counters in parameter summary

In the final report, there are several counters that should be understood:

- **Total number of reads**—the total number of reads in the 1-N BAM files for the sample. If the sample contains more than one BAM file, this number will be the total number of reads for all files.
- **Number of valid reads**—the number of remaining reads after bp-length cutoff and primer trimming.
- **Number of reads ignored**—any unique read with a copy number less than the set threshold will be removed. This number will then be the total number of reads removed.
- **Mapped reads in sample**—the number of reads that contained a valid result after BLAST mapping.
- **Unmapped reads in sample**—the number of reads that did not contain a result after BLAST mapping, plus the number of reads that had a result, but are not valid due to low coverage. Low coverage is when the mapped-against-result sequence is much shorter than the query sequence. The coverage percentage can be set in the user settings, and defaults to 90%. For example, for a query sequence that is 300 bp long, the returned mapped-against-result sequence should be at least 270 bp.

Counters in primer table

Some counters are the same as for the entire sample, but they are broken down per primer. These are not listed below:

- **# of forward found**—number of reads where the forward primer was found.
- **# of forward full coverage**—number of reads where both forward and reverse primers were found.
- **# of forward short**—number of reads where the forward primer (or both) was found, but the trimmed read did not meet the bp-length requirement.
- **# of forward valid reads**—number of valid reads after trimming, depending on whether just forward primer or both were needed.
- **# of reversed found**—number of reads where the reverse primer was found in the beginning of the read (a reversed-direction read).
- **# of reversed full coverage**—number of reversed reads where both reverse primer as forward primer and the reversed inverted forward primer as reverse primer were found.
- **# of reversed short**—number of reversed direction reads where the reverse primer as forward primer was found (or both), but the trimmed read did not meet the BLAST length requirement.
- **# of reversed valid reads**—number of valid reversed reads after trimming, depending on whether the forward primer or both were needed.

Numerical values in result tables

In addition to the taxonomy information, the tables also contain the following information:

- **% ID**—the range of percent ID match of the reads that gave the result.
- **Count**—the number of reads for this result line. This number is aggregated into the count number for the above taxonomy level.
- **DB counters**—number of reads for this result matched in the DBs used. If the result has 1,000 reads per count, and 900 of these were matched in DB #1 and 100 in #2, the information listed would be 900:100. If more DBs are used for multistage lookup, the additional number will be listed, separated by a colon (:).
- **F:R %**—ratio of forward and reversed reads for this result.
- **% of total reads**—the percentage of the total number of reads that this result makes up.
- **% of valid reads**—the percentage of the number of valid reads that this result makes up.
- **% of mapped reads**—the percentage of the number of mapped reads that this result makes up.

Section 4

Database and library files

Source file format

Any FASTA file can be used as source for a database generation. What matters most is that the header contains correct information.

In case of this algorithm, the correct format is:

Base/minimum header info:

>mg|Genus|Species|
(sequence)

Extra if available:

>mg|Genus|Species|Subspecies/
Strain|Accession#|Kingdom|Phylum|Class|Order|Family|PubMed#|LibraryID#|
(sequence)

Command-line command to generate DB

BLAST command line requires the database or library to be in a compatible format. The user can convert any FASTA file into this format by running the following command:

makeblastdb -in <source.fasta> -dbtype nucl -out <libraryfilename>

The resulting files needs to be placed into an accessible folder or directory by the BLAST command-line tool.

Cleaning up the new MSID DB

A custom utility was created that did the following:

1. Downloaded updated node information from National Technology for Biotechnology Information (NCBI)'s main FTP servers. This node information contains the complete hierarchy of all NCBI-registered references. Finding the node of a species makes it possible to traverse the tree up to the top node—via all taxonomy levels.

2. Downloaded updated naming information from NCBI's main FTP servers.

3. Extracted all scientific names or correct taxonomy names and matched those to the correct nodes.

4. Queried NCBI live to convert all Accession IDs in the library to Tax IDs.

5. Updated all taxonomy names for all levels above genus, where existing taxonomy name was different from updated NCBI information. This is done by traversing from bottom up (as described in step 1).

6. Generated new library source files ready for conversion into BLAST-compatible format.

Cleaning up Greengenes in Ion Repoter 5.0 workflow

The publicly available Greengenes library is a relatively large library containing more than 1.2 million references. As Greengenes is mostly aimed towards taxonomy information for family level and above, taxonomy information for genus and species levels were missing for almost 1 million references. A custom utility was used to curate Greengenes for genus- and species-level information by querying NCBI.

A custom utility was created that queried NCBI live to get the latest genus and species information:

1. Read Greengenes accession number information.
2. Read Greengenes taxonomy information.
3. Saved the references with valid genus and species information.
4. Queried NCBI live with batches of Greengenes accession numbers to find updated information on nearly 1 million references in Greengenes that did not contain any information on genus and species. (This process has been modified to run against local taxonomy files, which may be downloaded from NCBI).
5. The result was then filtered, and the references with new and updated genus and species names were added to the library created in step 3. This filtering step has been modified to remove entries where GenBank™ ID was missing for the corresponding Greengenes entry. However, if a Greengene entry contains all names for kingdom to species, it is retained.
6. References missing genus and species names that received an uncultured or unidentified flag instead of updated information were completely omitted.
7. The final library contained 215,967 references directly from Greengenes, and 146,822 references with updated genus and species names from NCBI, totaling 362,789 references.
8. This library was then converted into a BLAST-compatible format.

ion torrent

Find out more at **thermofisher.com/ion16S**

ThermoFisher SCIENTIFIC

CO020707 0116