HUMAnN: The HMP Unified Metabolic Analysis Network
Harvard School of Public Health

Author: Curtis Huttenhower
Version: 1.00
Effective Date:

1 Abstract

The human body is inhabited by trillions of bacteria and other microbes, which have recently been studied in many different habitats (including gut, mouth, skin, and urogenital) by the Human Microbiome Project (HMP). These microbial communities were assayed using high-throughput DNA sequencing, but it can be challenging to determine their biological functions based solely on the resulting short sequences. To reconstruct the metabolic activities of such communities, we have developed HUMAnN, a method to accurately infer community function directly from short DNA reads. The method's accuracy was validated using a collection of synthetic microbial communities.

2 Introduction

This SOP describes HUMAnN, a pipeline used for efficiently and accurately determining the presence/absence and abundance of microbial pathways in a community from metagenomic data.

Sequencing a metagenome typically produces millions of short DNA/RNA reads. HUMAnN takes these reads as inputs and produces gene and pathway summaries as outputs:

- The abundance of each orthologous gene family in the community. Orthologous families are groups of genes that perform roughly the same biological roles. HUMAnN uses the KEGG Orthology (KO) by default, but any catalog of orthologs can be employed with minor changes (COG, NOG, etc.)
- The presence/absence of each pathway in the community. HUMAnN refers to pathway presence/absence as "coverage," and defines a pathway as a set of two or more genes. HUMAnN uses KEGG pathways and modules by default, but again can easily be modified to use GO terms or other gene sets.
- The abundance of each pathway in the community, i.e. how many "copies" of that pathway are present.

HUMAnN can thus be used in tandem with any translated BLAST program to convert sequence reads into coverage and abundance tables summarizing the gene families and pathways in a microbial community. This lets you analyze a collection of metagenomes as a matrix of gene/pathway abundances, just like you might analyze a collection of microarrays.

Author: Curtis Huttenhower
Version: 1.00
Effective Date:

3 Requirements

3.1 Data requirements

One or more tabular translated BLAST (blastx) output files matching sequence read IDs to gene IDs. Place (or symlink) each file with a .txt, .txt.gz, or .txt.bz extension in the "input" directory before running HUMAnN. The pipeline includes processors for three tab-delimited text formats by default (below) and can easily by modified to accept more.

As an example, the default inputs provided with HUMAnN were generated using:
blastx -outfmt 6 -db 28_kegg_genomes < mock_even_lc.fasta.gz | gzip -c > mock_even_lc.txt.gz

Where "28_kegg_genomes" is a database of the amino acid sequences of 28 well-characterized KEGG organisms' ORFs (.pep files from ban, bbr, bqu, bsu, cbo, cdf, cje, eco, efa, ftu, hin, hpy, hsl, lmo, mbo, mtu, ngo, nme, pae, rso, sau, sco, sgo, spn, vch, xfa, yen, and yps).

3.2 Software requirements

HUMAnN depends on the following items:

- A network connection
  HUMAnN downloads a number of data and software components from standard repositories (primarily KEGG) during exection. Please ensure that a network connection is available at the least for the first run. HUMAnN will use "curl" by default to download files, and this can be changed (e.g. to use "wget") by editing the humann.py file.
- A bunch of RAM
  Processing one Illumina lane of metagenomic reads can take as much as ~8-10GB of memory, although it will often take much less depending on the data. Consider yourself warned.
- scons
  http://www.scons.org
- Python >= 2.7
  http://www.python.org
  As of this writing, we use exactly one feature unique to Python 2.7, math.gamma. If you're willing to edit it out, the dependency is to Python 2.6 for various modules and syntax.
- blastx
  http://www.ncbi.nlm.nih.gov/blast/
  Note that HUMAnN does not run blastx. It instead consumes tabular BLAST results as input. We recommend the default "-outfmt 6" setting as described below and in the

provided SConstruct configuration. Alternatively, input processors are also provided for accelerated BLAST implementations such as mapx and mblastx.

- MinPath (automatically downloaded)
  http://omics.informatics.indiana.edu/MinPath/
  See Ye et al, PLoS Computational Biology 2009
- KEGG (automatically downloaded)
  http://www.genome.jp/kegg/
  See Kanehisa et al, NAR 2010
- BioCyc (optional, automatically downloaded)
  http://biocyc.org
- maq (optional, automatically downloaded)
  http://maq.sourceforge.net
- R (optional, for synthetic performance evaluation)
  http://www.r-project.org
- R package ROCR (optional, for synthetic performance evaluation)
  http://rocr.bioinf.mpi-sb.mpg.de

4 Procedure

HUMAnN uses the scons build system to drive its scientific workflow (see *Requirements*). scons works very much like make, converting a set of inputs into a set of outputs one step at a time, and running only the steps necessary to produce the desired output.
HUMAnN is highly configurable in order to perform a collection of very computationally intensive tasks efficiently and flexibly. Please see the included sample input files, metadata files, and SConstruct settings for an overview of the software's configuration and the file formats it consumes and produces.

Use the following steps to analyze your data:

4.1 Populate the "input" directory

Place one or more translated BLAST results using KO identifiers in the "input" directory (optionally gzipped or bzipped).

Place (or symlink) each file with a .txt, .txt.gz, or .txt.bz extension in the "input" directory before running HUMAnN. The pipeline includes processors for three tab-delimited text formats by default (below) and can easily by modified to accept more.

As an example, the default inputs provided with HUMAnN were generated using:

```
blastx -outfmt 6 -db 28_kegg_genomes < mock_even_lc.fasta.gz | gzip -c > mock_even_lc.txt.gz
```

Where "28_kegg_genomes" is a database of the amino acid sequences of 28 well-characterized KEGG organisms' ORFs (.pep files from ban, bbr, bqu, bsu, cbo, cdf, cje, eco, efa, ftu, hin, hpy, hsl, lmo, mbo, mtu, ngo, nme, pae, rso, sau, sco, sgo, spn, vch, xfa, yen, and yps).

4.2 Edit the "SConstruct" file

In particular, make sure that the input processors include one configured for your BLAST file name(s) and format(s). Modify the SConstruct file as follows to specify the exact format of your input data:

blastx -outfmt 6

```
qseqid sseqid pident length mismatch gapopen qstart qend sstart send
evalue bitscore
```

mapx

```
template-name frame read-name template-start template-end template-length
read-start read-end read-length template-protein read-protein alignment
identical %identical positive %positive mismatches raw-score bit-score e-
score
```

mblastx

```
Query_Id Reference_Id E_Value_Bit_Score Identity_Percentage_Length_of_HSP
Number_of_Positives_Frame_#s Alignment_Start_in_Query
Alignment_End_in_Query Alignment_Start_in_Reference
Alignment_End_in_Reference
```

4.3 Run the "scons" command, optionally parallelizing multiple analyses using the "-j" flag. Results will be placed in the "output" directory.

By default, the following output file types are produced:

File	Description
_00-.txt.gz	Generated from raw BLAST results. A condensed binary representation of translated BLAST results, abstracted from and independent of the specific format (blastx/mblastx/mapx) in which they are provided.
_01-.txt	Generated from *_00-*.txt.gz. Relative gene abundances as calculated from BLAST results in which each read has been mapped to zero or more gene identifiers based on quality of match. Total weight of each read is 1.0, distributed over all gene (KO) matches by quality.
_02a-.txt	Generated from *_01-*.txt. Relative gene abundances distributed over all pathways in which the

	gene is predicted to occur.
_02b-.txt	Generated from *_02a-*.txt. Relative gene abundances with pathway assignments, taxonomically limited to remove pathways that could only occur in low-abundance/absent organisms.
_03a-.txt	Generated from *_02b-*.txt. Relative gene abundances with pathway assignments, smoothed so that zero means zero and non-zero values are imputed to account for non-observed sequences.
_03b-.txt	Generated from *_03a-*.txt. Relative gene abundances with pathway assignments, gap-filled so that gene/pathway combinations with surprisingly low frequency are imputed to contain a more plausible value.
_04a-.txt	Generated from *_03b-*.txt. Pathway coverage (presence/absence) measure, i.e. relative confidence of each pathway being present in the sample. Values are between 0 and 1 inclusive.
_04b-.txt	Generated from *_03b-*.txt. Pathway abundance measure, i.e. relative "copy number" of each pathway in the sample. On the same relative abundance scale (0 and up) as the original gene abundances _01-*.txt.
_99-.txt	Generated from *_00-*.txt.gz. Per-sample gene abundance tables formatted for loading into METAREP. On the same relative abundance scale (0 and up) as the original gene abundances _01-*.txt.
04a-*.txt	Combined pathway coverage matrix for all samples.
04b-*.txt	Combined pathway abundance matrix for all samples, normalized per column.
_01-keg.txt	Gene abundance calculated as the confidence (e-value/p-value) weighted sum of all hits for each read.
_02-mpt*.txt	Gene to pathway assignment performed using MinPath.
_02-nve*.txt	Gene to pathway assignment performed naively using all pathways.
_02-cop*.txt	Pathway abundances adjusted based on A) taxonomic limitation and B) the expected copy number of each gene in the detected organisms.
_03a-wbl*.txt	Smoothing performed using Witten-Bell discounting, which shifts sum_observed/(sum_observed + num_observed) probability mass into zero counts and reduces others by the same fraction.
_03a-nve*.txt	Smoothing performed naively by adding a constant value (0.1) to missing gene/pathway combinations.

_03b-nul*.txt	No gap filling (no-operation, abundances left as is).
_03b-nve*.txt	Gap filling by substituting any values below each pathway's median with the median value itself.
_04a-nve*.txt	Pathway coverage calculated as fraction of genes in pathway at or above global median abundance.
_04a-xpe*.txt	Pathway coverage calculated as fraction of genes in pathway at or above global median abundance, with low-abundance pathways set to zero using Xipe.
_04b-nve*.txt	Pathway abundance calculated as average abundance of the most abundant half of genes in the pathway.

Optional Components

You can reproduce the entire performance evaluation using synthetic communities in the HUMAnN manuscript with the tools in the "synth" subdirectory. This optional step is omitted during default HUMAnN operation but will be automatically incorporated into HUMAnN output if one or more synthetic communities are built. To do this:

- Place a .fa/.qual file pair or .fastq file in the "synth/output" directory and edit the "synth/SConstruct" file accordingly. This can be any representative sequencing data from which an error model will be built with which to synthesize artificial reads.
- Unfreeze the synthetic data by changing c_fFrozen to False in SConstruct. Run "scons" and wait quite a while. High-quality genomes will be downloaded from KEGG, shredded into artificial reads using maq, and mixed into a synthetic community.
- By default, four communities are built: two with staggered organismal abundances (stg, using a lognormal distribution) and two with even abundances (even), and two with 20 organisms (low complexity, lc) and two with 100 (high complexity, hc).
- The true gene and pathway abundances/coverages will be placed in correspondingly named files in the "synth/output" directory. These will be automatically detected the next time HUMAnN is executed using scons. They will be merged into the overall output files, and if correspondingly named input files ("mock_stg_lc*", "mock_even_hc*", etc.) are available, their performance will be automatically plotted as PDF output files using R.

Detailed Operation

HUMAnN's scons workflow runs on the assumption that each input file will be converted into one or more output files by combining a series of pipeable processing modules. It is also set up to allow easy configuration and comparison of output settings, so think of it as a tree: any output processor that _can_ process an input file _will_. This allows, for example, the quick

and easy comparison of the results of metabolic reconstruction using many slightly different parameter settings to determine which is optimal.

Note that while HUMAnN uses primarily KEGG Orthology KO identifiers for genes and KEGG pathway (ko) or module (M) identifiers for pathways, this can be easily modified to suit your analysis needs. For example, code to process MetaCyc reaction identifiers (for gene families) and pathway identifiers is also included by default. Any traceable identifiers (ECs, NOGs, etc.) can be used for genes and for the pathways in which they're contained.

The SConstruct file includes a configurable set of processing modules used to convert each input file into one or more outputs. Processing modules are defined using the CProcessor class with seven arguments:

1. A file suffix (for input files) or type (for output files). This can be of the form ".txt.gz" for input files or "##" for output files. This pattern must match the filename _consumed_ by the processor.
2. A file type. This is of the form "##" by default and can be any short tag (numerical or text) used to identify a type of generated file. File types are used to chain processing modules together; files of type ## will only and always be further processed by modules that can input type ##.
3. An identifier. This is of the form "xyz" by default and can be any short tag (numerical or text) used as a human-readable identifier for the processing module and its specific configuration options. It is appended to the resulting output file and provides a minimal form of data provenance.
4. A program. This is typically a Python script that consumes an input file on stdin, produces output on stdout, and can optionally include additional command line arguments (see below).
5. Zero or more supporting files as command line arguments. This array of file names is passed to the processing program as command line arguments (and will be understood by scons as dependencies, thus rebuilding the output when they change).
6. Zero or more additional command line arguments. This array of arbitrary strings is passed to the processing program as command line arguments (and will be ignored by scons). This can be used to produce multiple output files from the same processing scripts using command line arguments.
7. A boolean flag, True if the processor should be used for initial input files and False (or omitted) otherwise. In the latter case, the processor will only be used on intermediate files matching its input type.
8. A boolean flag, True if the output should be gzipped and False (or omitted) otherwise. Subsequent processors will automatically ungzip compressed input files.

Author: Curtis Huttenhower
Version: 1.00
Effective Date:

5 Implementation

An implementation of our methodology is available at http://huttenhower.sph.harvard.edu/humann. This provides a means to accurately and efficiently characterize microbial metabolic pathways and functional modules directly from high-throughput sequencing reads, enabling the determination of community roles in the HMP cohort and in future metagenomic studies.

6 Discussion

7 Related Documents & References

Curtis Huttenhower et al. "Metabolic reconstruction of microbial communities from metagenomic data". Unpublished.

8 Revision History

Version	Author/Reviewer	Date	Change Made
1.00	Curtis Huttenhower	09/20/2011	Establish SOP