Metagenomic Protein Function Prediction using the SFLD and Thresholded Sequence Similarity Networks

by

Jack Yu

THESIS

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

Biological and Medical Informatics

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, SAN FRANCISCO

[redacted]

Metagenomic Protein Function Prediction using SFLD and Thresholded Sequence Similarity Networks

Jack Yu, Patricia C. Babbitt

Abstract

The Structure-Function Linkage Database (SFLD) is a database containing hierarchical classifications of enzymes that relates specific sequence-structure features to specific chemical properties. It contains a collection of tools and data for investigating sequence-structure-function relationships and hypothesizing function. Currently, users can query one or more “unknown” protein sequences against the database using Hidden Markov Model or BLAST, and be able to compare, classify, annotate against existing curated enzyme *superfamilies*, the largest grouping of proteins for which common ancestry can be inferred. Here we present a working pipeline that allows users to putatively assign functions to sequences derived from metagenomics studies and to visualize relationships between these sequences and existing enzyme superfamilies using thresholded sequence similarity networks.

Table of Contents

List of Figures

Introduction

Our world is home to a diverse assemblage of microbial species. While cultivation-independent methods employing PCR-amplification, cloning and sequence analysis of 16s rRNA or other phylogenetically informative genes have made it possible to assess the composition of microbial species in natural elements, until recently this approach has been too time consuming and expensive for routine use. Advances in high throughput metagenomics sequencing have largely eliminated these obstacles, reducing cost and increasing sequencing capacity by orders of magnitude[1].

Shotgun metagenomics sequencing is an alternative approach to the study of uncultured microbiomes. But, instead of targeting a specific genomic locus for amplification, all DNA is subsequently sheared into tiny fragments that are independently sequenced. This results in DNA sequences (i.e., reads) that align to various genomic locations for the myriad genomes present in the sample, including non-microbes. Some of these reads will be sampled from taxonomically informative genomic loci (e.g., 16S), and others will be sampled from coding sequences that provide insight into the biological functions encoded in the genome[2]. However, for our analysis, we will begin with the translated version of these reads, also known as protein sequences.

The accurate annotation of protein function is key to understanding life at the molecular level. However, with its inherent difficulty and expense, experimental characterization of function cannot scale up to accommodate the vast amount of sequence data already available. The computational annotation of protein function has therefore emerged as a problem at the forefront of bioinformatics. Recently, the availability of genomic-level sequence information for thousands of species, coupled with massive high-throughput experimental data, has created new opportunities as well as new challenges for function prediction. Many methodologies have been developed by research groups worldwide, many based in comparing unsolved sequences with databases of proteins whose functions are known. Other methods aim at mining the scientific literature associated with some of these proteins, yet others combine sophisticated machine-learning algorithms with an understanding of biological processes to decipher what these proteins do[3]. In the Babbitt Lab, we use the first of the aforementioned method: using SFLD's well-curated known proteins to predict unknowns through the utilization of sequence similarity networks.

Methods

<u>Objective</u>:

The objective of this project is to use the SFLD to create hypotheses about the functions of protein sequences derived from translated metagenomic reads and to make putative assignments of these proteins within the boundaries of specific enzyme superfamily and subgroups. Finally, we aim to visualize these assignments against the valid members of the superfamily/subgroup to produce sequence similarity networks (Figure 1).

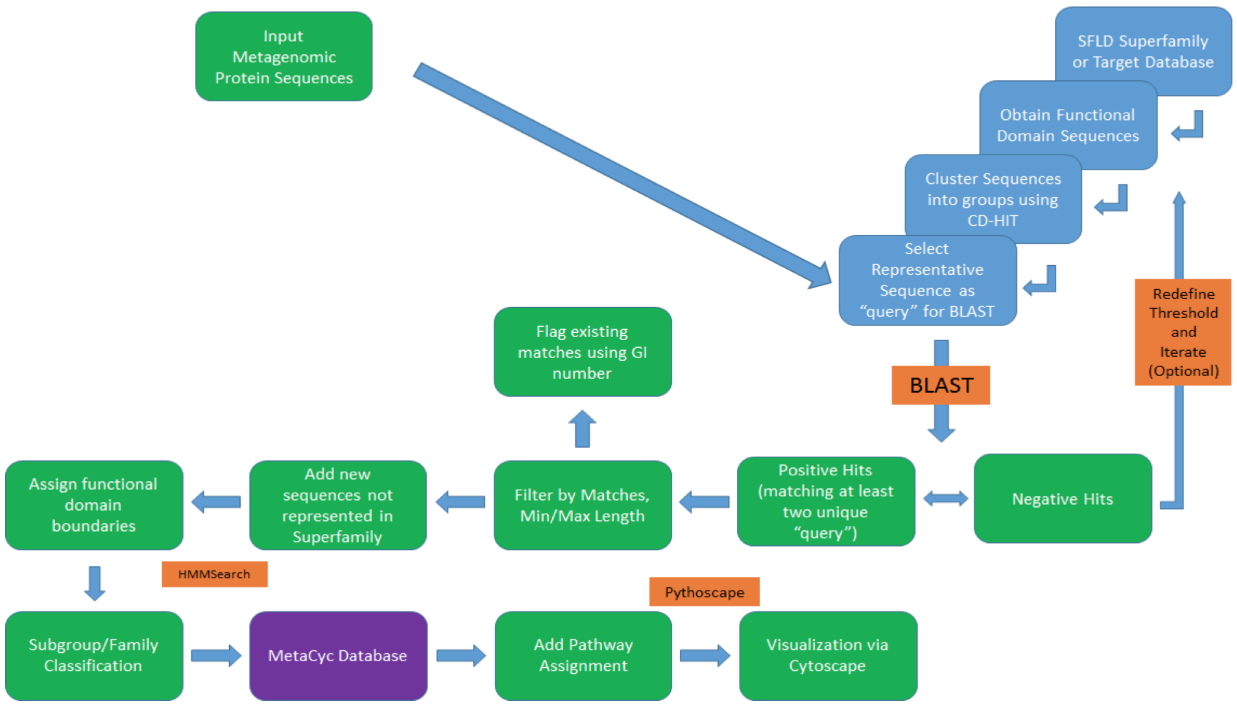

Input Metagenomic Protein Sequences
SFLD Superfamily or Target Database
Obtain Functional Domain Sequences
Cluster Sequences into groups using CD-HIT
Select Representative Sequence as "query" for BLAST
Redefine Threshold and Iterate (Optional)
Flag existing matches using GI number
BLAST
Assign functional domain boundaries
Add new sequences not represented in Superfamily
Filter by Matches, Min/Max Length
Positive Hits (matching at least two unique "query")
Negative Hits
HMMSearch
Pythoscape
Subgroup/Family Classification
MetaCyc Database
Add Pathway Assignment
Visualization via Cytoscape

Figure 1. Proposed Pipeline

Input:

The input of this pipeline is a set of unknown metagenomics protein sequences in the form of a FASTA file, as well the SFLD Superfamily functional domain and annotation files to which the unknown files may match (based on user preference). Additionally, subgroup multiple sequence alignment (flat) files are used to assign sequences to individual subgroups.

Output:

The output this pipeline produces is a XGMML file using the Pythoscape[7] that can be visualized using Cytoscape 3.2.1. It also produces summary statistics for the analyzed sequences including their enzyme superfamily and subgroup assignment as well as the overall count of the number sequences successfully assigned to an individual category.

Dependencies/Tools:

CD-HIT v4.6.3[4], a widely used program for clustering and comparing protein or nucleotide sequences. It is very fast and can handle extremely large databases which makes this suitable for metagenomics-based analyses. CD-HIT helps to significantly reduce the computational and manual efforts in many sequence analysis tasks and aids in understanding the data structure and correct the bias within a dataset.

BLAST 2.2.30[5], stands for basic local alignment search tool, an algorithm for comparing pairwise primary biological sequence information, such as amino-acid sequences of different proteins or the nucleotides of DNA sequences. A BLAST search enables us to compare a query sequence within a library or database of sequences, and identify library sequences that resemble the query sequence above a certain threshold. Here we are going to use Protein Blast (blastp) for searching protein databases (SFLD) using protein query sequences (the "unknown" set).

HMMER v3.1b2[6]**,** a tool used for searching sequence databases for homologs of protein sequences, and for making protein sequence alignments. It implements methods using probabilistic models called profile hidden Markov models[10]. Compared to BLAST, FASTA and other sequence alignment and database search tools based on older scoring methodology, HMMER aims to be significantly more accurate and more able to detect remote homologs because of the strength of its underlying mathematical models. HMMER is now as fast as BLAST.

Pythoscape v1.04[7], a framework for generation of large protein similarity networks created by Dr. Alan Barber of the Babbitt Lab. Protein similarity networks are graphical representation of sequence, structural and other similarities among proteins for which pairwise all-by-all similarity connection have been calculated. Mapping of biological and other information to network nodes or edges enables hypothesis creation about sequence-structure-function relationships across sets of related proteins.

Pandas 0.16.1[8], is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools that is used in the script.

Pipeline Breakdown

For a given superfamily, all functional domain sequences, i.e. the portion of the sequence relevant to the superfamily are obtained from the SFLD database. They are then filtered by validity, keeping only those sequences with a record having a NULL "valid_until" and a non-empty superfamily assignment evidence code. These sequences are subsequently clustered into groups of 40% identity using CD-HIT, the lowest threshold supported by the tool, but it is sufficient for our application of collapsing the number of sequences via clustering to a manageable and meaningful number for downstream analyses. We refer to these groups as representative sequence groups. The longest sequence is selected by default as the representative sequence for each group (subject to change based on prior knowledge of the query).

All of the representative sequences are then grouped to be the query for a BLAST search against the unknown protein sequences at a pre-defined (but can be altered if prior knowledge dictates) e-value of 1e^{-10} with SEG filtering and soft masking on. SEG is a tool that finds areas of low compositional complexity, for example regions of biased amino acid composition like histidine-rich domains, so using the filter improves the output sequence quality. Soft masking is another procedure that identifies low-complexity sequences, such as repeats like ATATATATATAT or regions that are highly enriched for just one letter, e.g. AAACAAAAAAA. Protein segments with only a few amino acids are also considered to be low complexity and while they could align with a high score, it would not necessarily indicate an evolutionary relationship.

The output of the BLAST procedure produces a file containing all of the unknown sequences that meet the criteria of the parameters previously defined. We can putatively define these sequences to be part of the enzyme superfamily. Next step is to assign functional domains to the new sequences, as well as classify the new sequences into subgroups. To achieve this, we must first find the single closest SFLD subgroup for each newly assigned sequence by using HMMER. The reason why HMMER is used over BLAST in this instance is due to the reason that sometimes BLAST can potentially pick functionally unrelated proteins containing promiscuous domains, whereas HMMER will examine only the domains of interest. Because we use functional domains to characterize subgroup assignment, HMMER is the preferred method for this application. Another way to combat the BLAST caveat would be to restrict the match length (i.e. the subject should match at least 80% of the query protein). Two procedures of the HMMER tool will be used here: hmmbuild and hmmsearch. Hmmbuild builds a profile HMM from a multiple sequence alignment file, which is one of our inputs and obtained from subgroup pages on the SFLD, under the “View Alignment” tab. This HMM profile is then used as a benchmark to be searched against, and the alignments of the profile HMM to the best-scoring sequences are displayed in the output.

When it comes to classifying the new sequences to the subgroup, two steps must be taken to test for subgroup membership. First, hmmsearch is used to query the subgroup HMM against the valid family members (obtained from SFLD, under “Download Data Set” tab), and the lowest matching domain bit score is calculated as the threshold by which the unknown sequence must pass in order to be assigned to the subgroup. Then the unknown sequences follow the same procedure to produce bit-scores of their own. If the closest family bit score is smaller than the lowest bit score of a valid family member, then the family assignment is dropped for this particular unknown sequence. This process is iterated until all of the BLAST hits have either been assigned or dropped from the queue. The output from the hmmsearch include total number of query sequences analyzed and the subset of these that are reported over the threshold that are previously defined for each of the subgroups.

After the subgroup assignment, visualization is performed using Pythoscape, which undergoes the following steps. First, sequences are added to the network as nodes using UniprotKB regular expression to parse the sequence tags and call the identifiers “Uniprot”. Note that different sequence files will require different regular expression for file header parsing. Edges are subsequently added to the network, along with a list of edge attributes desired such as metrics like bit score, alignment length, -ln(E) and –log10(E) values, as well as alignment identities, query start and query end. Uniprot information is subsequently imported (contingent upon having BioPython installed prior to Pythoscape), and a representative node network is created for the purpose of a simplified view and reduced visualization object size. The representative node sizes are then calculated and applied, as well as the representative node edges. Finally, upon importing representative node attributes, the representative node network is outputted, along with the full node network.

The subsequent output is in the form of .xgmml and can be imported directly into Cytoscape, under File -> Import -> Network -> File. The import process will depend on the size of the input file; it could take up to several minutes for larger files. When the network is first opened, the view is rather uninformative. To resolve this, lay the network using one of the many layout options Cytoscape provides, here we choose to use yFiles -> Organic Layout. This layout presents the nodes as squares, each representing a single sequence (or a representative sequence in the case of a representative network), and the connecting edges between the nodes represent BLAST connections that are at least as significant as the default e-value cutoff of $1e^{-20}$, or if another e-value cutoff is specified previously during the network generation process. In this layout, the edges between sequences represent only connectivity. They show the two nodes are connected at the specified e-value cutoff. While the length of the edges do not directly represent how distant these proteins are from each other, they do give a reasonable indirect approximation. Selecting individual node or edge will display the imported annotations performed by the Pythoscape procedure. Users will have the ability to color code the nodes and edges, as well as removing them based on certain criteria within the Cytoscape tool.

Sample Use/Preliminary Results

The input used to test our pipeline is a curated list of query protein sequences (332,809) from the Koenig[11] data that are no shorter than 40 amino acids in length. The reason for such threshold is that the BLAST engine is principally designed for searching sequences over full-length sequences, and searching the databases with short sequences may cause error using available BLOSUM matrices. The origin of this dataset is from previous work where we looked to study the metabolic pathway distribution in the infant microbiome.

In the longitudinal study conducted by Koenig *et al.*[11], the authors obtained sixty fecal samples from an infant boy over the course of two and a half years to examine the in-depth dynamics of a developing intestinal ecosystem. Using 16s rRNA data, the authors found that the phylogenetic diversity of the microbiome increased gradually over time, and seemingly chaotic shifts in the microbiota composition are directly associated with real-life perturbations. By studying the landscape of the gut Metagenome surrounding illnesses, antibiotic administrations, and dietary changes (available in Koenig data) we can theorize not only how these events impact the species diversity within the gut, but also some of the functions of those proteins that are present in these species. Our assumptions are that gut species interactively (symbiotically and competitively) perform necessary functions for survival. Dysbiosis (disruption of the gut microbiome equilibrium) by external perturbations can lead to the extermination of species (along with their functions), and the gut landscape will reflect such changes.

To test our proof of concept model, I selected two superfamilies of interest to analyze: Enolase, one of the first and most extensively curated superfamily in the SFLD; and Glutathione Transferase (cytosolic). The Enolase Superfamily proteins share the core chemical step of an abstraction of a proton from a carbon adjacent to a carboxylic acid and a requirement of a divalent metal ion. In contrast to many recognized families of enzymes whose members catalyze similar reactions on different substrates, the enolase superfamily includes enzymes catalyzing a wide variety of reactions and performing diverse roles in metabolism. The Glutathione Transferase (GST) superfamily proteins are major phase II detoxification enzymes found mainly in the cytosol, where their role in catalyzing the conjugation of electrophilic substrates to glutathione is their primary function.

Using our pipeline, we performed analyses on both superfamilies. Of the 332,809 unknown metagenomics sequences analyzed, 3,253 (nearly one percent) were mapped to the Enolase superfamily. Of which, 131 sequences were assigned to the Enolase subgroup, 8 to Galactarase, 33 to Glucarate, 63 to Mandalate racemase, 31 to Manoate, 12 to Methylaspartate and 39 to Muconate. The remaining 3,122 sequences could not be confidently assigned to any of the subgroups. Altogether, this set of unknown sequences contributed to 317 new sequences to many of the subgroups within the Enolase superfamily. On the other hand, 8,991 sequences of the unknown query were mapped to the GST Superfamily, 2.7 percent coverage. However, only ten sequences were assigned to individual subgroups: two for the AMPS: Alpha-, Mu-, Pi-

and Sigma-like subgroup and eight for the Main subgroup. This astronomical difference in superfamily and subgroup assignment could be the result of a highly variable subgroup: low-confidence cytGST-like proteins, which vary significantly in functional domains and do not have a public alignment available while occupying more than 38 percent of all of Glutathione Transferase Superfamily. (Pending figure for visual validation, network generation on-going and will complete by 06/09/15, figure will be added for final document submission).

Discussion/Significance

Comparing the metagenomics data to the existing SFLD will give us a better perspective of how prevalent and similar the enzymes with well-characterized functions in human compare to the enzymes that exist within a microbial community. These microbial communities range from the human gut, to fresh water Lake, to soil from the Tibetan Plateau, to everyday consumer products such as kimchi and artisanal cheese, they exist all around us. Adding these new metagenomics data can potentially elucidate evolutionary relationships between enzymes that catalyze reactions (or share functional domains) in known pathways from different origins.

This is a proof-of-concept model for incorporating Metagenomic sequences using existing protein databases and functional prediction protocols. The statistics obtained and the visualization step will provide insight as to how prevalent the enzymes with well-characterized functionality (core-SFLD superfamilies) are present in the “unknown” metagenomics ecosystems, and they can also help us identify trends in which these enzymes have evolved by comparing how the similar sequences diverged whilst in different environments.

This pipeline offers users, initially within the Babbitt Lab, and eventually to a broader audience interested in studying the functions of proteins and translated metagenomic reads, significant flexibility in usage for different types of projects. The code will be open source and made public online for users to modify and apply. In addition to the built-in features such as modifiable e-values thresholds and superfamily selection, we also offer the flexibility to analyze a single sequence versus joining multiple sequence files of interest into one unified filed for analyses using the –single or –full arguments. The user is prompted this choice at the command-line so that it will reduce the amount of time for data processing. This pipeline also allows users to create protein sequence similarity networks of reasonable size for the protein groups of their choice with ease. Validation is done by visualization Cytoscape, to map the assigned unknown sequences to the enzyme superfamily and verify that the assignment is correct/useful based on pairwise similarity. This pipeline allows for significant flexibility (target superfamilies and subgroups using powerful existing resource of SFLD, similarity cut-offs using e-values and flexible cd-hit clustering criteria, network customization and visualization using Pythoscape), throughout the entire process of beginning with a set of unknown queries, to final output of a sequence similarity network. The Babbitt Lab is currently upgrading the Metagenomics Network construction protocol, thus the current state of this pipeline may yield incomplete network output. For larger size (nodes or edges) networks, computation on local machines may be laborious, use clusters as an alternative if possible.

Future Directions

This proof-of-concept is just the beginning of many different ways that metagenomics data can be explored within the context of the SFLD. Discovering how these well-curated enzymes are distributed in various metagenomes is simply the first step to learn how similar or different the human proteome is compared to those of the hundreds of trillion microbes that exist on Earth. A next step could be to explore the enzyme/pathway space over time, for example in metabolism of the human gut.

Changes in everyday life activities can have profound impact on the gut microbiome function. Creating an inventory of metabolic profile of an individual containing enzymes, reaction and pathway information could be tremendous for the study of the temporal colonization patterns of the gut microbes and how they contribute to human metabolism. One way to do so is to use the publically available resource of Pathway Tools[9] created by Peter Karp. Pathway Tools is a comprehensive systems-biology software that is closely associated with BioCyc/MetaCyc database (Figure 2).

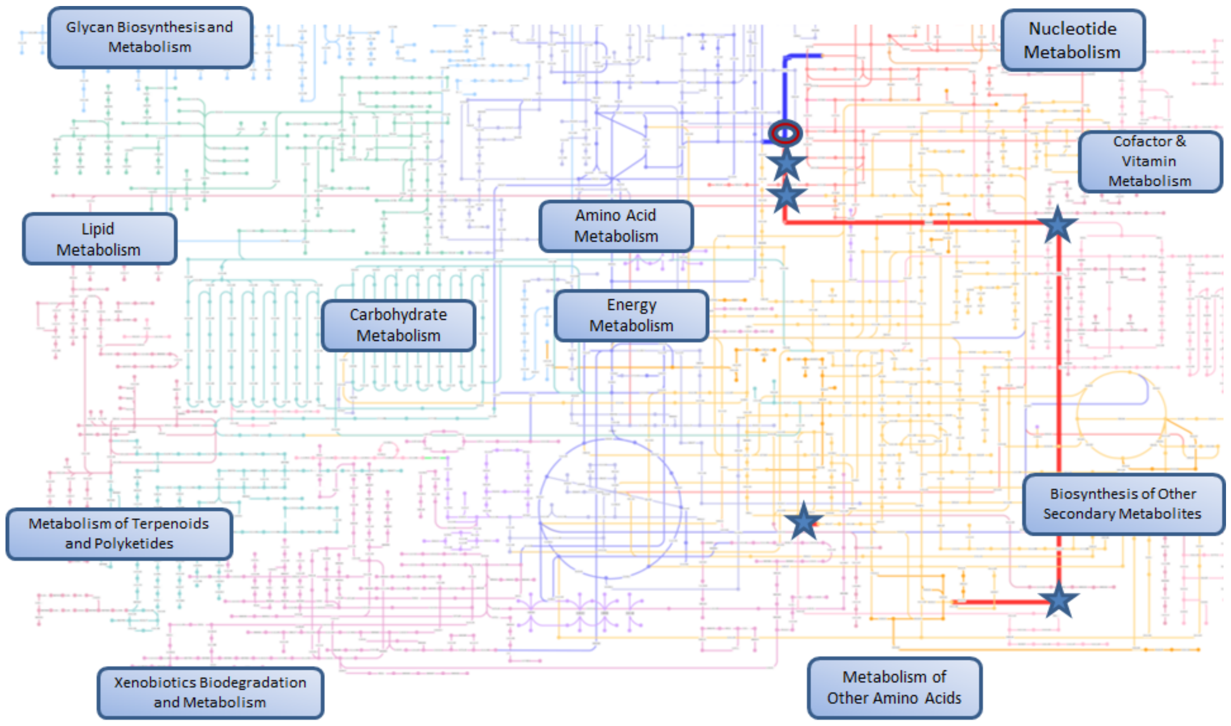
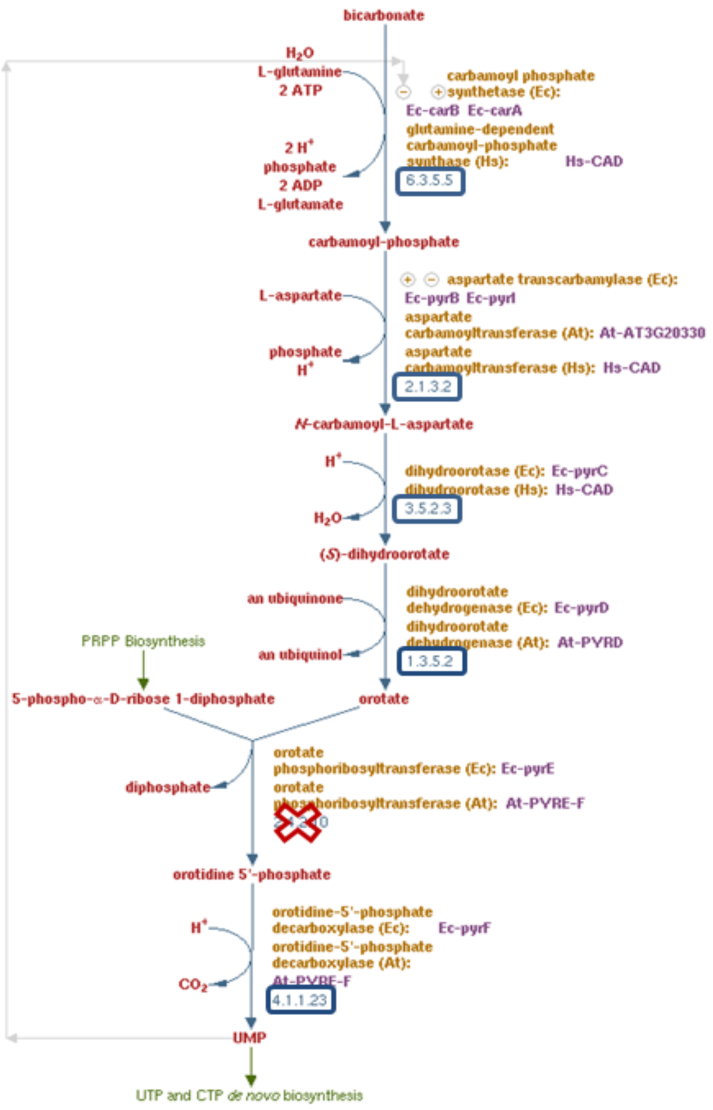

Glycan Biosynthesis and Metabolism
Nucleotide Metabolism
Cofactor & Vitamin Metabolism
Lipid Metabolism
Amino Acid Metabolism
Carbohydrate Metabolism
Energy Metabolism
Biosynthesis of Other Secondary Metabolites
Metabolism of Terpenoids and Polyketides
Xenobiotics Biodegradation and Metabolism
Metabolism of Other Amino Acids

Fig. 3 Reference visualization of metabolic pathway profile based on E.C. number/reactions; Stars along the red line denote the members of the Pyrimidine Biosynthesis Pathway; The circle (2.4.2.10) and blue line denote a singular enzyme in the pathway that is not present in all time points from the Koenig data. Isolated star is E.C. 4.2.2.10, which is not connect to the pathway *via* KEGG resource, but is documented by MetaCyc as part of the pathway. Previous work by J.Yu.

It has expanded from 371 to over 5500 well defined pathway/genome databases since 2009 and is currently one of the best resources for enzyme/protein pathway information today.

Given that we have information about origin of the enzymes and what reactions they catalyze, adding pathway-relevant information would contribute immense context to understanding the evolution of these enzymes (Figure 3).



bicarbonate
H_2O
L-glutamine
2 ATP
carbamoyl phosphate synthetase (Ec):
Ec-carB Ec-carA
glutamine-dependent carbamoyl-phosphate synthase (Hs): Hs-CAD
2 H^+
phosphate
2 ADP
L-glutamate
6.3.5.5
carbamoyl-phosphate
L-aspartate
aspartate transcarbamylase (Ec):
Ec-pyrB Ec-pyrI
aspartate carbamoyltransferase (At): At-AT3G20330
aspartate carbamoyltransferase (Hs): Hs-CAD
phosphate
H^+
2.1.3.2
N-carbamoyl-L-aspartate
H^+
dihydroorotase (Ec): Ec-pyrC
dihydroorotase (Hs): Hs-CAD
H_2O
3.5.2.3
(S)-dihydroorotate
an ubiquinone
dihydroorotate dehydrogenase (Ec): Ec-pyrD
dihydroorotate dehydrogenase (At): At-PYRD
PRPP Biosynthesis
an ubiquinol
1.3.5.2
5-phospho-α-D-ribose 1-diphosphate
orotate
orotate phosphoribosyltransferase (Ec): Ec-pyrE
orotate phosphoribosyltransferase (At): At-PYRE-F
diphosphate
orotidine 5'-phosphate
H^+
orotidine-5'-phosphate decarboxylase (Ec): Ec-pyrF
orotidine-5'-phosphate decarboxylase (At): At-PYRE-F
CO_2
4.1.1.23
UMP
UTP and CTP de novo biosynthesis

Fig. 3 Pyrimidine Biosynthesis Pathway via MetaCyc Database, blue box indicates that the enzyme is present in the metagenomics data, red cross indicates the enzyme is absent. Data from previous work on the Koenig study.

One way to approach the linkage between enzymes and pathways could be through the usage of Enzyme Commission reference numbers, which is a numerical classification scheme for enzymes based on chemical reactions they catalyze. As a system of enzyme nomenclature, every E.C. number is associated with a recommended name. Each enzyme has a set of four numbers that define the classes and sub-classes of enzyme it belongs to with each number giving a more specific definition than last. The attribute of E.C number is chosen because of it directly determines whether or not the unknown protein is likely to be an enzyme based on sequence similarity (Figure 2).

Another direction that we could pursue given our model is to compare microbial metagenomes against one another, rather than against the known database, and track the movement of the enzyme presence using full sequence similarity networks. This approach could be applied to study time-lapsed data or target-specific perturbation based ecological studies where significant portion of the microbial community will remain constant, and only a certain subset of the population will be affected, thus the migration of enzyme presence from the norm can be detected. For example, to study the proteomic diversity of ocean water over during different seasons, or microbial life in snake and spider venoms, or shifts in human intestinal microbiota after smoking cessation. The data for all of the above are currently available in the EBI Metagenomics data bank.

```
#!/usr/bin/python

import sys, os

if len(sys.argv) == 1:
    print ''
    print 'Function: Blast a fasta file against another'
    print ''
    print 'Usage: blast_FaToFa [-full OR -single] [sequenceA Directory]
[sequenceB.fasta] [sequenceB type: n (nucleotide) p(protein)] [blast program: blastn,
tblasx, blastp, blastx, tblastn] [evalue] [output]'
    print ''
    sys.exit()

if len(sys.argv) > 1 and len(sys.argv) < 6:
    print 'incorrect number of parameters'

typeParse = sys.argv[1]
queryDir = sys.argv[2]
dbFile = sys.argv[3]
baseType = sys.argv[4]
blastProgram = sys.argv[5]
evalue = sys.argv[6]
outFile = os.path.abspath(os.path.dirname(sys.argv[7])) + "/" +
sys.argv[7].split('/').pop()

if typeParse == '-full':
   l00 = os.listdir(queryDir)
   l01 = [k for k in l00 if ".faa" in k]

   file_t = open(queryDir + "/queryFile.fasta", 'w')
   l1 = list()
   for i in l01:
      tmp = open(queryDir + "/" + i)
      tmp00 = tmp.readlines()
      tmp01 = [j.replace("\n","") for j in tmp00]
      l1.extend(tmp01)


   file_t.write("\n".join(l1))
   file_t.close()
   queryFile = queryDir + "/queryFile.fasta"

else:
   l00 = os.listdir(queryDir)
   l01 = [k for k in l00 if ".faa" in k]
   print "Select file:"
   print ""
   print "\n".join(l01)
   print ""
   query = raw_input("File to BLAST: ")
   queryFile = queryDir + "/" + query


if baseType == 'p':
    baseType = 'prot'
elif baseType == 'n':
    baseType = 'nucl'
else:
    print 'wrong parameter for base type'
    sys.exit()

makedbcmd = 'ncbi-blast-2.2.30+/bin/makeblastdb.exe -in ' + dbFile + ' -dbtype ' +
baseType + ' -title temp -out tempDB'

blastdbcmd = 'ncbi-blast-2.2.30+/bin/blastp.exe -query ' + queryFile + ' -db tempDB -
evalue ' + evalue + ' -outfmt 6 -out blast.out -num_threads 4 -max_target_seqs 20'

blastindexcmd = 'ncbi-blast-2.2.30+/bin/blastdbcmd.exe -db tempDB -dbtype ' + baseType
+ ' -entry all -out tempDB.out'

print 'making blast database'
os.system(makedbcmd)
print 'blasting fasta files'
os.system(blastdbcmd)
print 'creating index'
os.system(blastindexcmd)

os.system('grep "^>" tempDB.out | tr -d ">" > tempDB2.out')

index = dict([(line.strip().split(' ',1)[0],line.strip().split(' ',1)[1]) for line in
open("tempDB2.out",'r').read().strip().split('\n')])

blastResult = open('blast.out','r')
finalFile = open(outFile,'a')

for line in blastResult:
    data = line.split('\t')
    data[1] = index[data[1]]

    finalFile.write('\t'.join(data) + "\n")

blastResult.close()
finalFile.close()


# NOTE: you may want to check results of BLAST before running the next sequence

# Sequence 2: parsing data into FASTA files for visualization
import pandas as pd, re

# CUSTOMIZE
# set output path depending on local setup
path = 'output1'

# read in sequences from BLAST
txt = open("tempDB.out")
l1 = txt.readlines()

# CUSTOMIZE
# read in superfamily index from SFLD download
ind = pd.DataFrame.from_csv("sfld_superfamily_19.tsv", sep = "\t")


seq = pd.DataFrame(columns = ("orig", "efdid", "uniprot", "subgroup"))

# find domain and uniprot ID and grouping into table 'seq'
for i in range(0,len(l1)):
   if ">" in l1[i]:
      efd = re.search("EFDID\|(.*)\|", l1[i]).group(1)
      seq.loc[i] = [l1[i],efd,ind[ind.index ==
int(efd)].uniprot.values[0],ind[ind.index == int(efd)].subgroup.values[0]]


# group into domain sequence
os.makedirs(path)
```

```
for i in list(set(seq.subgroup)):
   if str(i) == "nan":
      continue
   print i

   #os.makedirs('/users/chluo/Downloads/FASTA temp/'+str(i))
   tmp = seq[seq.subgroup == i]
   tmp_list = list()
   for j in range(0,tmp.shape[0]):
      idx = l1.index(tmp.orig.values[j])
      l1_tmp = l1[idx+1:len(l1)]
      if str(tmp.uniprot.values[j]) == "nan":
         continue
      print tmp.uniprot.values[j]
      for k in range(0, len(l1_tmp)):
         if ">" in l1_tmp[k]:
            tmp_list.extend(">sp|"+tmp.uniprot.values[j]+"\n")
            tmp_list.extend(l1[(idx+1):(idx+k+1)])
            print k+1
            break

   file_t = open(path + "/" + str(i) + '.fasta', 'w')
   file_t.write("".join(tmp_list))
   file_t.close()

# outputting all sequences in one FASTA file
l00 = os.listdir(path)
l01 = [k for k in l00 if ".fasta" in k]
txt00 = list()
for i in l01:
   txt = open(path + "/" + i, "r")
   txt00.extend(txt.readlines())
txt_w = open(path + "/full_sequence.fasta", "w")
txt_w.write("".join(txt00))
txt_w.close()

# output master table of sequence mappings
seq.to_csv(path+"/master table.csv")

def count(seq, pred):
    return sum(1 for v in seq if pred(v))


txt2 = open(queryFile, 'r')
txt2_dat = txt2.readlines()
txt2.close()
file_t = open('blast.out','r')
file_t_txt = file_t.readlines()
file_t.close()

# output summary metrics
print str(len(file_t_txt)) + " sequences mapped out of " + str(count(txt2_dat, lambda
i:  ">" in i)) + " at e-value of " + str(sys.argv[6])

exit()

# NOTE: you may want to check if FASTA files are too large and only keep ones under
200kb for visualization (potential memory-related problem)

# Sequence 3: Visualization

# Import Pythoscape
import pythoscape.main.environments as env
import pythoscape.interface.local_interface as l_i
from pythoscape.auxiliary.re_patterns import RE_PATTERNS


# Import plug-ins
import pythoscape.plugin.input.import_sequences as i_s
import pythoscape.plugin.input.add_local_blast as a_l_b
import pythoscape.plugin.input_bio.add_uniprot_info as a_u_i
import pythoscape.plugin.input.make_cdhit_repnodes as m_c_r
import pythoscape.plugin.input.repnode_stats as r_s
import pythoscape.plugin.input.add_repnode_edges as a_r_e
import pythoscape.plugin.input.add_repnode_atts as a_r_a
import pythoscape.plugin.output.output_xgmml as o_x
import pythoscape.plugin.input.add_attribute_table as a_a_t



files = os.listdir(path)
files = [i for i in files if "fasta" in i]
for i in range(0,len(files)):
   # Create interface and environment
   os.makedirs(path + "/" + files[i].replace(".fasta",""))
   my_interface = l_i.LocalInterface(path + "/" + files[i].replace(".fasta",""))
   my_pytho = env.PythoscapeEnvironment(my_interface)
   my_net = env.PythoscapeNetwork('rep-net',my_interface)



   #Create plug-ins

   #Import sequences
   plugin_1 = i_s.ImportFromFastaFile(path + "/" +
files[i],id_re=RE_PATTERNS['UniprotKB'],id_name='Uniprot')

   # Input edges to Pythoscape
   # CUSTOMIZE
   plugin_2 = a_l_b.AddBLASTEdgesFromLocalBLAST("ncbi-blast-
2.2.30+/bin/blastp.exe",include_atts=['-log10(E)'])

   #Input edges to Pythoscape
   plugin_3 = a_u_i.ImportFromUniProt(uniprot_id='Uniprot')

   #Create representative network
   # CUSTOMIZE
   plugin_4 = m_c_r.CreateCDHITRepnodes('rep-net',"cd-hit-v4.6.1-2012-08-27/cd-
hit",c=0.7,n=5)

   #Create representative network
   plugin_5 = r_s.CalcNodeSize('rep-net','cdhit 0.7','node size')

   #Create representative network
   plugin_6 = a_r_e.AddEdgesToRepnodeNetwork('rep-net','cdhit 0.7','-
log10(E)',filt_dir='>',filt_value=0)

   #Create representative node attributes
   plugin_7 = a_r_a.AddAttributesByIfAny('rep-net','cdhit 0.7','family','repnode
family')

   #Join Attribute Table
      # CUSTOMIZE
   # read in superfamily index from SFLD download
   plugin_8  =
a_a_t.ImportAttributeTable('sfld_superfamily_19.tsv',match_against='existing_identifer
')

   #Output network
   out_plugin_1 = o_x.OutputXGMML(path+"/"+files[i].replace(".fasta","")+'/gst-sigma-
rep-net-20.xgmml','sigma gst representative node network @ 20',filt_name='rep-net
mean',filt_dir='>',filt_value=20)

   #Output network
   out_plugin_2 = o_x.OutputXGMML(path+"/"+files[i].replace(".fasta","")+'/gst-sigma-
20.xgmml','sigma gst network @ 20',filt_name='-log10(E)',filt_dir='>',filt_value=20)

   #Execute plug-ins
   my_pytho.execute_plugin(plugin_1)
   my_pytho.execute_plugin(plugin_2)
   my_pytho.execute_plugin(plugin_3)
   my_pytho.execute_plugin(plugin_4)
   my_pytho.execute_plugin(plugin_5)
   my_pytho.execute_plugin(plugin_6)
   my_pytho.execute_plugin(plugin_7)
   my_pytho.execute_plugin(plugin_8)
   my_net.execute_plugin(out_plugin_1)
   my_pytho.execute_plugin(out_plugin_2)

   print str(files[i])+ " visualization complete"
```

```
import sys, os

if len(sys.argv) == 1:
    print ''
    print 'Function: hmmsearch across a group of sequences'
    print ''
    print 'Usage: hmm.py [querySeq] [alignment Directory] [benchmark Directory]'
    print ''
    sys.exit()

if len(sys.argv) > 3 or len(sys.argv) == 2:
    print 'incorrect number of parameters'

queryDir = sys.argv[1]
dbDir = sys.argv[2]
benchDir = sys.argv[3]

l00 = os.listdir(dbDir)
for i in l00:
   # Make sure hmmer executable is installed
   os.system("hmmer-3.1b2-cygwin64/binaries/hmmbuild " + dbDir + "/" + i + ".build " +
dbDir + "/" +i)

#creating benchmark
l00 = os.listdir(dbDir)
l01 = [k for k in l00 if ".build" in k]
for i in l01:
   os.system("hmmer-3.1b2-cygwin64/binaries/hmmsearch " + dbDir + "/" + i + " " +
benchDir + ">" + dbDir + "/" + i + ".output")

#running against queryDB
for i in l01:
   os.system("hmmer-3.1b2-cygwin64/binaries/hmmsearch " + dbDir + "/" + i + " " +
queryDir + ">" + dbDir + "/" + i + "_query.output")


l00 = os.listdir(dbDir)
l01 = [k for k in l00 if "_query.output" in k]
for i in l01:
   txt = open(dbDir + "/" + i)
   l1 = txt.readlines()
   print i.replace(".build_query.output", "") + " Summary Metrics"
   print ""
   print "".join(l1[l1.index("Internal pipeline statistics summary:\n"):len(l1)])
```

References

1. A framework for analysis of metagenomic sequencing data. A. Murat Eren, Michael J. Ferris, Christopher M. Taylor Pac Symp Biocomput. 2011 : 131–141.
2. Sharpton TJ. An introduction to the analysis of shotgun metagenomic data.*Frontiers in Plant Science*. 2014;5:209. doi:10.3389/fpls.2014.00209.
3. Radivojac P, Clark WT, Oron TR, Schnoes AM, Wittkop T, Sokolov A, Graim K, Funk C, Verspoor K, Ben-Hur A, Pandey G, Yunes JM, Talwalkar AS, Repo S, Souza ML, Piovesan D, Casadio R, Wang Z, Cheng J, Fang H, Gough J, Koskinen P, Törönen P, Nokso-Koivisto J, Holm L, Cozzetto D, Buchan DW, Bryson K, Jones DT, Limaye B, Inamdar H, Datta A, Manjari SK, Joshi R, Chitale M, Kihara D, Lisewski AM, Erdin S, Venner E, Lichtarge O, Rentzsch R, Yang H, Romero AE, Bhat P, Paccanaro A, Hamp T, Kaßner R, Seemayer S, Vicedo E, Schaefer C, Achten D, Auer F, Boehm A, Braun T, Hecht M, Heron M, Hönigschmid P, Hopf TA, Kaufmann S, Kiening M, Krompass D, Landerer C, Mahlich Y, Roos M, Björne J, Salakoski T, Wong A, Shatkay H, Gatzmann F, Sommer I, Wass MN, Sternberg MJ, Škunca N, Supek F, Bošnjak M, Panov P, Džeroski S, Šmuc T, Kourmpetis YA, van Dijk AD, ter Braak CJ, Zhou Y, Gong Q, Dong X, Tian W, Falda M, Fontana P, Lavezzo E, Di Camillo B, Toppo S, Lan L, Djuric N, Guo Y, Vucetic S, Bairoch A, Linial M, Babbitt PC, Brenner SE, Orengo C, Rost B, Mooney SD, Friedberg I.
   A large-scale evaluation of computational protein function prediction (2013) Nature Methods 10, 221–227
4. Limin Fu, Beifang Niu, Zhengwei Zhu, Sitao Wu and Weizhong Li, CD-HIT: accelerated for clustering the next generation sequencing data. Bioinformatics, (2012), 28 (23): 3150-3152. doi: 10.1093/bioinformatics/bts565. PDF, pubmed.
5. Camacho C., Coulouris G., Avagyan V., Ma N., Papadopoulos J., Bealer K., & Madden T.L. (2008) "BLAST+: architecture and applications." BMC Bioinformatics 10:421. PubMed
6. *HMMER web server: interactive sequence similarity searching*
   R.D. Finn, J. Clements, S.R. Eddy
   Nucleic Acids Research (2011) Web Server Issue 39:W29-W37. PDF
7. Alan E. Barber II and Patricia C. Babbitt
   Pythoscape: a framework for generation of large protein similarity networksBioinformatics (2012) 28 (21): 2845-2846 first published online September 8, 2012doi:10.1093/bioinformatics/bts532
8. Wes McKinney pandas: a Foundational Python Library for Data Analysis and Statistics; presented at PyHPC2011
9. [PTools10] P.D. Karp, S.M. Paley, M. Krummenacker, *et al*
   "Pathway Tools version 13.0: Integrated Software for Pathway/Genome Informatics and Systems Biology,"
   Briefings in Bioinformatics 11:40-79 2010.
10. Hidden Markov Model Speed Heuristic and Iterative HMM Search Procedure. L. S. Johnson, S. R. Eddy, E. Portugaly. *BMC Bioinformatics*, 11:431, 2010.
11. Succession of microbial consortia in the developing infant gut microbiome. Koenig JE[1], Spor A, Scalfone N, Fricker AD, Stombaugh J, Knight R, Angenent LT, Ley RE. Proc Natl Acad Sci U S A. 2011 Mar 15;108 Suppl 1:4578-85. doi: 10.1073/pnas.1000081107. Epub 2010 Jul 28.