Xander: Gene Targeted Metagenomics

Jordan A Fish*[1,2], Yanni Sun[2] , James M Tiedje[1], James R Cole[1]

[1]Center for Microbial Ecology, Michigan State University [2]Department of Computer Science and Engineer, Michigan State University

Email: Jordan Fish*- fishjord@msu.edu; Yanni Sun - yannisun@msu.edu; James M Tiedje - tiedjej@msu.edu; James R Cole - colej@msu.edu;

*Corresponding author

Abstract

Background: Metagenomics can provide important insight in to microbial communities. It can be used to analyze entire genomes and takes full advantage of increasing sequencing capacity. However analyzing large metagenomic datasets has proven to be very computationally challenging with even modest metagenomic datasets requiring 256 gigabytes of memory or more (cite hmp assembly, titus). As dataset size progresses from 50 gigabases to 500 gigabases and beyond new methods are needed for deriving understanding from metagenomic datasets. We present a method for assembling protein coding sequences for gene(s) of interest from a metagenomic dataset which uses a compressible graph format and only assembles targeted data to drastically reduce the amount of memory and processing time required.

Results: ...

Conclusions: ...

Background

Metagenomics has the potential to help answer many questions but has faced scalability challenges stemming from the amount of raw sequencing data required for metagenomic analysis. Metagenomic

assembly has been an area of interest in recent years with the early datasets assembled using single genome assembly approaches. The tendency for single genome assemblers to only assemble a few dominant organisms has been an impetus to develop metagnomic specific assembly methods. Currently metagenomic assembly approaches focus on separating individual organism genomes out of the metagenomic data to be worked on individual (Metavelvet, Partitioning).

We propose a gene targeted approach for assembling metagenomic datasets: HMMgs. HMMgs is a De Bruijn Graph (cite) assembler but unlike traditional approaches that find either a Eulerian or Hamiltonian path through the graph, we use a protein profile Hidden Markov Model (HMM) (cite) to target specific paths in the graph and guide assembly. By using an HMM we can explore most probable paths first to limit the amount of the graph we must explore along with obtaining a measure of how well the resulting assembled sequence is from the supplied model. Using this gene targeted approach allows for a functional based analysis of the data,by allowing us to directly examine genes involved in biologically interesting pathways.

Gene targeted assembly is less resource intensive and faster than whole genome assembly. In addition to the De Bruijn Graph, only small paths relative to the graph's size must be kept in memory. Further reduction in the memory usage are achieved by using a probabilistic data structure for holding the De Bruijn Graph in memory, a Bloom filter [1] (cite kmer percolation). By targeting relatively small segments of the metagenome by using an HMM to guide assembly, we limit how much of the graph must be explored during assembly, providing a speed up over whole genome approaches.

Our Method is more sensitive than whole genome assembly and more specific than individual read-based approaches fro functional analysis. By using an assembly approach we have more context from which to make a determination as to whether a stretch of sequence belongs to the target gene or not, increasing our sensitivity compared to read based approaches. Using HMM probabilities to guide local assembly helps to ensure we explore the most relevant paths to assemble sequences most likely to represent the gene of interest, increasing our sensitivity compared to uniform global assembly.

We use a graph search based approach to assemble the targeted genes. We create a graph structure on demand that is a combination of a De Bruijn Graph and HMM. A De Bruijn Graph is a graph where vertices represent fixed K-length words (K-mers) and edges link vertices that share K-1 word. A HMM can also be thought of as a graph in which each vertex represents a model state and edges represent allowed transitions between states. By combining the two we create a graph structure that can be easily searched using existing graph search algorithms.

Results and Discussion

We tested HMMgs on three approx. 50 gigabase Illumina GAII soil metagenomes: two Great Prairie Challenge Iowa datasets, Iowa Corn and Iowa Prairie, and one Miscanthus Rizosphere set from a GLBRC plot at the Kellog Biological Station. We focused on Nitrogen Reductase (nifH) and 50S ribosomal protein L2 rplB for these datasets. NifH is important in nitrogen fixation and expected to be found in the datasets albiet at low levels. The rplB gene coding for ribosomal protein L2 is present in all bacteria and archaea and was used to benchmark our program on highly abundant genes.

Preparing Genes of Interest

For each gene of interest we selected a set high-quality seed sequences to build a HMM model and a larger set used to identify possible starting Kmers. Seed sequences were obtained from the Ribosomal Database Project's Functional Gene Repository (FGR) (cite) [How were these selected?]. Sequences for the larger reference set were also downloaded from the FGR. All sequences that covered at least 90% of the HMM and with a good bits saved score, dependent on the gene, were selected as the larger reference set.
We used the seed sequences to build an HMM for each gene using a modified version of HMMER3 (cite). While testing HMMgs we found that HMMER3's default delete and insert prior probabilities appeared much higher than seen in the genes we were examining an such high priors sometimes caused extensive searching of nonproductive insert and delete paths. with our metagenomic data. We modified HMMER3's source code to change the prior probabilities for the $delete \rightarrow match$ and $insert \rightarrow match$ to 95% probability, $delete \rightarrow delete$ and $insert \rightarrow insert$ to 5% probability. Our modifications to HMMER3 are available as a patch file for version 3.0.

Processing

Each dataset was quality filtered by trimming reads at quality score "B" as recommended by Illumina [2]. A Bloom filter was built for each dataset on Amazon EC2 using a quadruple extra-large instance with hash size of 32 gigabytes, K-mer size of 30, using 4 hash functions and a bitset size of 16. Building the Bloom filters using one thread required ca. 8 hours for each dataset.
Starting points were identified in each of the datasets by using the protein references for nifH and rplB using an Amazon EC2 quadruple extra large instance using two threads and required 2 hours for each dataset. We ran HMMgs to search each dataset for nifH then rplB using a timing cutoff of 30 seconds per path extension. The results were filtered keeping hits with bits saved ≥ 50.

Since our starting point heuristic is not perfect some starting K-mers may be from reads coming from non-target genes. In this situation all paths in the combined graph are low scoring which causes the A* search to break down in to a full graph search, an exponential search. To identify these non-target K-mers we use a search time cutoff as almost all good starting K-mer searches terminate quickly.

NifH

Our nifH model had 42 seed sequences and a length of 296. The reference set contained 715 sequences. There was little similarity between any of the three samples using the jaccard similarity measure; however the corn and prairie samples were more similar to each other than to the miscanthus sample.

RplB

Our rplB model was built from 117 model and had a length of 277. The reference set contained 3573 sequences.

Conclusions

Since HMMgs is a gene targeted approach it requires some knowledge of what we are looking for; knowing what genes we are interested, we will not be able to identify novel proteins with a gene targeted approach. Our approach also requires up front effort in the form of selecting sequences as references and building a model for each gene of interest instead of doing the sequencing and then putting all the effort in to identifying protein coding regions after. This method also will not work for gene families that are too phylagentically diverse to be accurately modeled with an HMM.

While using HMMgs we identified some non-productive K-mers that did not produce results. We believe are K-mers shared across multiple domains and thus are identified for further analysis but do not produce high scoring paths. To deal with these problematic K-mers we added a maximum search time and any search that took longer than the maximum time was terminated. In the future we plan to implement methods to prune unproductive paths such as discarding paths that do not meet certain scoring criteria to speed up searching and detect unproductive paths without the need for a search time limit.

HMMgs's worst case processing time scales linearly with the number of starting vertices and exponentially with the length of the model. The actual processing time varies depending on the HMM state the starting vertex is in, the density of the graph around the starting vertex, and the time-out value specified to terminate non-productive searches.

Nifh

There was a low recovery of nifh sequences from the two Iowa samples. This is due in part to the lower estimated cover of the Iowa samples (5x) to the estimated coverage of the Miscanthus sample (10x). We also would not expect as much nifH in the corn sample as the field has been fertilized.

Methods

Graph Structure

We implemented a graph structure that combined a De Bruijn Graph (DG) and HMM together in a single combined graph (CG). A vertex in CG is defined for every pair of vertices u, v in DG and HMM:

$$\forall(u, v)\, u \in \text{DG},\, v \in \text{HMM}$$

each vertex in CG combines the information in u and v. The total number of vertices in CG will be

$$|V(DG)| * |V(HMM)|$$

where V(G) is the vertex set of the graph G, however vertices in CG can be generated as needed to reduce the memory requirements.

We defined the edge set of CG, E(CG), formally; suppose w_i, and $w_j \in V(CG)$ and were made by combining vertices v_i with u_i and v_j with u_j respectively with v vertices from the De Bruijn Graph and u vertices from the HMM.

$$\overrightarrow{w_i w_j} \in E(\text{CG}) \leftrightarrow \overrightarrow{v_i v_j} \in E(\text{DG}) \text{ and } \overrightarrow{u_i u_j} \in E(\text{HMM})$$

. That is, an edge exists in CG if and only if an edge connects the vertices they were created by combining. The weights of an edge $\overrightarrow{uv}$ in CG are the sum of the transition and emission probabilities taken from the HMM.

$$w(\overrightarrow{uv}) = P_{transition}(u \to v) + P_{emission}(v)$$

The emission symbol is the unique character in the K-mer contained in v.

Protein De Bruijn Graphs

In practice, we built our De Bruijn Graph in nucleotide space regardless of whether the HMM is modeling protein or nucleotide sequences. When searching with a protein HMM we treat any vertex in the nucleotide De Bruijn Graph that is exactly three steps in the same direction from the current vertex as a

neighbor in the inferred Protein De Bruijn Graph. The emission symbol then becomes the three unique characters at the end of the K-mer translated to protein. The codon reading frame is fixed based on the vertex chosen to begin graph traversal.

Identifying Search Starts

We used a K-mer matching heuristic to identify K-mers in the experimental reads that were present in a set of reference sequences for the gene of interest. The reference sequences were aligned to the HMM to allow us to assign the K-mer matches to a position on the HMM. The K-mer and the model position from the aligned reference, and implicit match HMM state were combined to form a vertex in CG.

To find overlapping K-mers we used an exact seed matching approach. The reference sequences were broken up in to K-mers and stored in a hash. Then we processed each read one at a time, looking up each K-mer in the read up in the hash of the references K-mers. For use with a protein HMM a seed length of $\lfloor K/3 \rfloor$ was used and input reads were translated in to all six reading frames (We limit K to a multiple of three for simplicity). When assembling multiple genes of interest the reference sets were combined together in to a single hash so potential search starts could be identified in a single pass over the sequence reads.

Searching

We used an implementation of the A* search algorithm [3] for finding paths through CG, modified to find the highest scoring path instead of the lowest cost path. We defined the set of goal vertices as any vertex in the last model position that is in the match or delete state. We define the scoring function for a path P as:

$$S(P) = \sum_{i=0}^{|P|} w(P_i P_{i+1})$$

where w(...) is the weight of the edge between two vertices in P

We define the heuristic cost function for a vertex v as:

$$h(v) = P_{v_{state} \rightarrow match} + \sum_{i=v_{modelposition}+1}^{M} P_{match \rightarrow match}(i, i+1)$$

the sum of the most likely state transitions from a v's state to the end of the model. Where P is the probability of the given transition and M is the length of the HMM.

The log-odds edge weights were transformed to ensure the heuristic score and scoring function were monotonic as follows:

$$w(\overrightarrow{uv}) = w(\overrightarrow{uv}) - max(P_{emission}(v_{HMMstate})) \quad u_{HMMstate} \neq i$$

Since this heuristic score will never overestimate the actual score it meets the admissibility criteria for A*, and additionally since the scoring function is monotonic a closed set was not required.
To deal with search starting positions beginning somewhere other than the beginning of the model we built both a forward and reverse model to enable searching in both directions from a starting vertex.

Multiple Paths

We implement a Kth Shortest Path algorithm [4] [5] to find multiple high scoring sequences from a single starting vertex. Yen's algorithm was modified so that if an edge had been seen in any of the K-1 shortest paths it was not considered in subsequent candidate generation iterations. This ensured that each next shortest path generated contained at least one new vertex.

Abundance

To determine how many copies of a contig assembled using HMMgs are present in a metagenome we used a K-mer counting method. A hash was built from all the K-mers in the assembled contigs. The sequence reads were then scanned looking each K-mer up in the contig hash, counting each time a contig K-mer was seen and the read it was in. With the occurrence information we were then able to compute both K-mer and base counts.

Implementation

HMMgs was implemented in the Java programming language and is distributed under a XXX License. We used a Bloom filter to store a compressed representation of the De Bruijn Graph. HMMgs consists of four primary tools; one for building a Bloom filter De Bruijn Graph, a tool for identifying starting positions, the HMMgs tool, a tool to merge left and right contig fragments, and a contig abundance counting tool.
Any of the tools can be replaced with a 3^{rd} party tool using difference heuristics as long as the resulting file matches the expected format. For example the starting vertex identification can be replaced with a 3^{rd} party tool so long as the resulting file contains the starting kmer and starting model position.

Sequence Analysis

Contigs generated by HMMgs were merged using the merger tool and filtered all sequences with bits saved less than 50 or a merged length of less than 100. Sequences were then aligned using HMMER3 and clustered using the RDP's Functional Gene Pipeline (cite). Beta diversity analysis was done by computing

the Jaccard similarity index with the Chao abundance correction (cite). R was used to generate heatmap visualizations of the data.

Authors contributions

Text for this section ...

Acknowledgements

Text for this section ...

References

1. Bloom BH: **Space/time trade-offs in hash coding with allowable errors**. *CACM* 1970, **13**(7):422–426.
2. Mann T: **Illumina Quality Scores** 2009, [https://docs.google.com/viewer?a=v&pid=explorer&chrome=true&srcid=0B-lLYVUOliJFYjlkNjAwZjgtNDg4ZC00MTIyLTljNjgtMmUzN2M0NTUyNDE3&hl=en_US].
3. Hart PE, Nilsson NJ, Raphael B: **A Formal Basis for the Heuristic Determination of Minimum Cost Paths**. *IEEE Transactions of Systems Science and Cybernetics* 1968, **4**(2):100–107.
4. Yen JY: **Finding the K Shortest Loopless Paths in a Network**. *Management Science* 1971, **17**(11):712–716.
5. Lawler EL: **A Procedure for Computing the K Best Solutions to Discrete Optimization Problems and its Application to the Shortest Path Problem**. *Management Science* 1972, **18**:401–405.

Figures

Figure 1 - Sample figure title

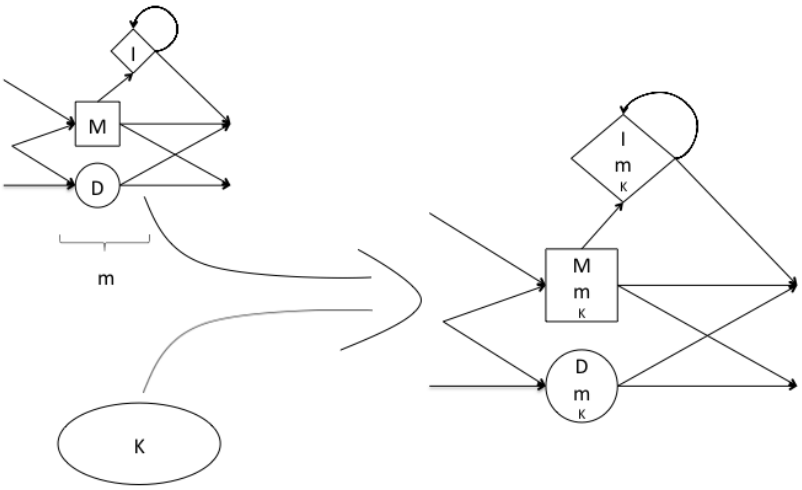
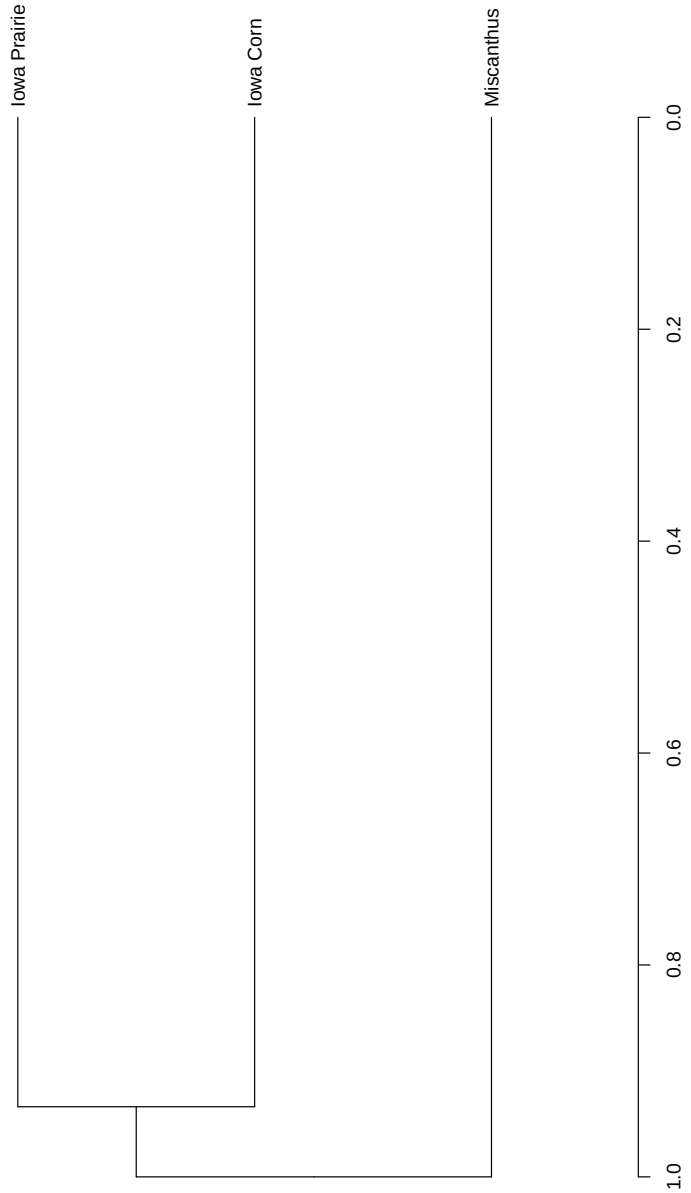


Figure 2



Tables

Table 1 - Sample table title

Here is an example of a *small* table in LaTeX using `\tabular{...}`. This is where the description of the table should go.

Dataset	Unique Starting vertices	Contigs Generated	Time (s)	Contigs merged	Unique
Iowa Corn	2169	4035	2481	73	3
Iowa Prairie	3472	6671	3268	922	62
Miscanthus Rizosphere	23788	46493	15487	2379	262

Table 2 - Sample table title

Large tables are attached as separate files but should still be described here.

Additional Files

Additional file 1 — Sample additional file title

Additional file descriptions text (including details of how to view the file, if it is in a non-standard format or the file extension). This might refer to a multi-page table or a figure.

Additional file 2 — Sample additional file title

Additional file descriptions text.