A Bayesian Probabilistic Assembler for Metagenomics

Computational Genomics Final Project Write Up

Farhan Damani, Dan Adler

December 2015

1

Abstract

Sequencing technology is used to generate a set of noisy DNA reads from a sample. In a metagenomics setting, the sample can have a high amount of species diversity. Assemblers built for single-sequence reconstruction are not sensitive enough to deal with this setting. We propose a Bayesian probabilistic assembler that incorporates prior information to help determine the most likely set of DNA sequences that would have generated the observed set reads. Our model is a simplified version of the Genovo assembler[1].

2

Introduction

Assembly for metagenomics demands our attention because it serves as a unique bottom-up approach to the discovery of new species. We can potentially discover new genomes simply by collecting data from unknown environments and attempting to assemble the reads we collected from that sample. An additional motivation is to discover how a virus such as a cancer cell or an HIV strain might evolve over time. For some viruses, the genetic diversity is related to its disease progression, which means this type of assembly can provide useful information for drug therapies.

This problem faces unique difficulties because we have no knowledge about what the reference genome(s) look like. More specifically, we have no knowledge about the population or the composition of its genomes. Our Bayesian probabilistic approach to assembly allows us to incorporate prior information to the model and accommodate for varied levels of sequence abundance, thus providing with us a more accurate set of likely reconstructed DNA sequences from the observed read set.

3

Prior Work

Laserson et al. built Denovo, a Bayesian probabilistic assembler for metagenomics. Their major selling point was to incorporate a nonparametric prior, accounting for the unknown number genomes in the sample. Their generative model was able to reconstruct sequences even for low-coverage species and was able to achieve better reconstruction performance compared to several state of the art short read assembly programs on real and synthetic data sets.[1]

Li-Thiao-Te et. al. attempt to quantity the total number of species in a genetically diverse environment such as a microbial sample. They focus on the situation where a large number of the species are not observed due to poor sampling of the community. They use a mixture model to estimate the total number of species. They also explore variational approximate inference techniques to perform inference instead of sampling methods such as Markov Chain Monte Carlo. This was useful for our project as we debated back and forth on what approximate inference techniques to use.[2]

Work by Zagordi et al. generated a probabilistic model for assigning observed reads to unobserved haplotypes using a Dirichlet process mixture for non-parametric clustering. They built a Gibbs sampler for sampling from the joint posterior distribution of haplotype sequences to learn about local population structures. They focused their work on deep sequencing data from HIV samples.[3]

Beyond specific literature on metagenomics, we spent a significant portion of our time learning about various approximate inference techniques that we thought might be useful, including Markov Chain Monte Carlo (MCMC)– metropolis hastings algorithm, variational inference techinques, and understanding how to combine probabilistic and deterministic steps into a coherent story. We read various tutorials on these topics online, consulted textbooks, and got help from Alexis Battle.

4

Methods and Software

A significant portion of our time on this project was spent developing the mathematical model for our probabilistic assembler. While we borrowed several ideas from the Laserson et al. paper, we incorporated several of our own assumptions that resulted in some significant changes to how our algorithm would work. We developed our own priors on variables, derived maximum likelihood estimations, explored various approximate inference techniques including various sampling techniques and chose Markov Chain Monte Carlo (the metropolis-hastings algorithm) in order to make joint probability computations tractable, and incorporated a noise model distribution based on the errors we inputted into our synthetic data set.

4.1 Data Collection

We downloaded three Illumina sequenced viruses: mp13mp18, papayameliera, and wuhanaphi. We created a synthetic dataset by taking all k-mers of length 100 from all of the fully sequenced viruses. We then shuffled the k-mers and replaced one percent of the bases with a random nucleotide to simulate a substitution error.

4.2 Design

Our algorithm is a random walk in the space of all potential assemblies and tells a generative story, discovering the most likely sequences the observed reads originated from. We first walk through the algorithm step-by-step. We then describe to you the generative model and the variables used. This is followed with a detailed description of the model.

```
randomly assign each read x_i to contig s, where s = 1,...alpha.
randomly assign each read x_i to a position o in contig s, where o =
    1,....l.

while likelihood function has not converged:
    read mapping
    consensus sequence
    if necessary, merge contigs
```

We place a geometric prior on the distribution over contigs and a uniform prior over the distribition of offset positions in a contig.

4.3 Assumptions

We assume the max number of contigs (α) is a set number. α is dependent on how many species we want to include in our model.

We assume the max length of any contig is 10k base pairs.

4.4 Variables

Observed variables:
x_i: vector of letters of read i

Unobserved variables:
y_i: alignment of read i
s_i: contig index of read i
o_i: starting offset of read i within contig s_i
b_{so}: DNA letter in offset o of contig s

Parameters
Q : number of reads
Q_s: number of reads in contig s
α: number of contigs (constant)
l: length of each contig (constant)
β: $\{A, C, G, T\}$

4.5 Likelihood

We seek to compute the total data log likelihood of all variables. The total data log likelihood reduces to:

$$log(p(x, y|s, o, b)) + logp(b) + logp(o|s) + logp(s)$$

4.6 Consensus Sequence

Consensus sequencing updates contig values based upon the overlapping reads at each offset, contig. We then build frequency distributions at each offset in a contig, and the most frequent nucleotide at each offset (the consensus) is chosen as the updated sequence value for that offset in a specified contig.

More precisely,

$$P(b_{so}|x, y, s, o, b_{so})$$

$$b_{so} = argmax_{b \in \beta} a^b_{so}$$

We break ties by choosing the nucleotide that appeared most frequently first.

4.7 Read Mapping

The read mapping step is the only probabilistic algorithmic step in our model. We seek to compute:

$$P(s_i = s, o_i = o, y_i = y | x, y\_i, s\_i, o\_i, b) \ \alpha$$

$$P(s_i = s | \{s\} - i) * P(o_i = o | s_i = s) * P(x_i, y_i = y | s_i = s, o_i = o, b)$$

To make sampling more tractable, consider:

$$y = y^*_{so} = argmax_y P(x_i, y_i = y | s_i = s, o_i, b)$$

.
Any optimal alignment algorithm will work here since we are only evaluating data sets with a substitution error. We computed y^* using a hamming distance function. Now, over all possible (s,o):

$arg\,max\ P(s_i = s, o_i = o | y^*, .)\ \alpha\ P(s_i = s | \{s\} - i) * P(o_i = o | s_i = s) * P(x_i, y^*_{so} | s_i = s, o_i = o, b).$

This is still too difficult to do exact inference on. We need a sampling procedure, so we will use Markov Chain Monte Carlo (MCMC). Specifically, Metropolis-Hastings, and will randomly sample without replacement 80 percent of the reads for this computation at each iteration.

For each iteration, these are the equations used to do the respective computations:

$$p(s) = (1-p)^{s-1} p, \ p = \frac{1}{Expected\ Contigs}$$

$$p(o) = \frac{1}{Contig\ Length}$$

$$p(x, y) = (1 - p_{miss})^{n_{hit}} p^{k - n_{hit}}_{miss},$$

where p_{miss} is the percent substitution across all reads, n_{hit} is the number of matches between x and y and k is the length of x_i.

This algorithm will output a new set of mappings for each read x_i, so for each x_i, we will have new values for y,s,o,and b.

4.8 Merge Contigs

The Merge Contigs algorithm is utilized to check whether after each iteration of the algorithm (i.e. a read mapping and consensus sequence) contigs overlap enough to be merged. In summary, each suffix and prefix of contigs is checked with each other contig to see whether there is overlap. We utilize the methodology described by Laserson to merge two contigs if there exists an overlap of greater or equal to than 15 nucleotides between the suffix of one contig and the prefix of another [1]. If there exists two contigs with prefixes that match a suffix of another contig, the larger of the two are kept.

After contigs have been merged, the reads must be mapped to their new position and offset in the contig list. Thus, the second part of the algorithm tracks where each contig mapped to, and at what offset the new mapping occurrence is, and then updates read $[s, o]$ values based upon this new mapping. More detail is described below:

4.8.1 Algorithm

Inputs and outputs to the merge contig algorithm are as follows:

Input: The current list of contigs and a read dictionary with keys as each read and values as lists of contigs, offsets, and probabilities of being at that contig/offset.
Output: A list of new contigs (merged) and new read mappings. If it was detected that no merge should occur, the original contigs and reads are returned.

The algorithm will be broken down into pieces:

We first run go through each pair of contigs and check if there exists an overlap between the suffix of the first contig and the prefix of the second greater than or equal to some threshold. As described, the default threshold is 15, but one can input a new threshold if wanted. Note that we need to check each pair twice, since, if given contig_1 and contig_2, once we check if the suffix of 1 overlaps the prefix of 2, we must also check if the suffix of 2 overlaps the prefix of 1. The algorithm then starts as follows:

```
1 Initialize dictionary for each contig with value as a Counter object =
    contig_dict
2
3 For each contig take suffix
4     For each other contig take prefix
5         store contig_dict[suffix][prefix] = suffix/prefix overlap if >=
      threshold
6
7 if there exists one overlap >= threshold, return contig_dict
8 else return []
```

Since it is often the case that no contigs will overlap on a given run, we keep track of whether there exists any relevant overlap. This is done using methodology for the previous homework assignment, by finding the best buddy to the left and right of each contig. Contigs are then traced to find overlap graph paths, which then merge

individual contigs together[4].

We store each path as a list, where each entry is another list that shows the path that contig makes in the overlap graph (i.e. the contigs to be merged). Each node in the path is a list of pairs stating the overlap of the nodes prefix to its previous element suffix and the element itself. This list is returned as a 'contig_trace', which is used to then overlap traced contig paths.

We need to change the read mappings to where the reads now fall within each contig set. This is done in two steps. In the first step we 'reverse' the read mapping data structure.

```
Create blank reverse dictionary = reverse_reads

for each read
    for each [s,o,p] triplet for that read
        add [read, o, p] to reverse_reads[s]

return reverse_reads
```

The final step in the algorithm is to then go through this reverse dictionary and create a new reads dictionary by taking the contigs path trace, the reverse reads and make a new reads dictionary by mapping each contig to its path index in the path trace (and this is the index it corresponds to in the new contigs list). We also need to input the original contigs list and also original reads dicitonary, because there might be reads that were not changed in the original mapping, and we make sure to include all reads and their new correct mapping in our output.

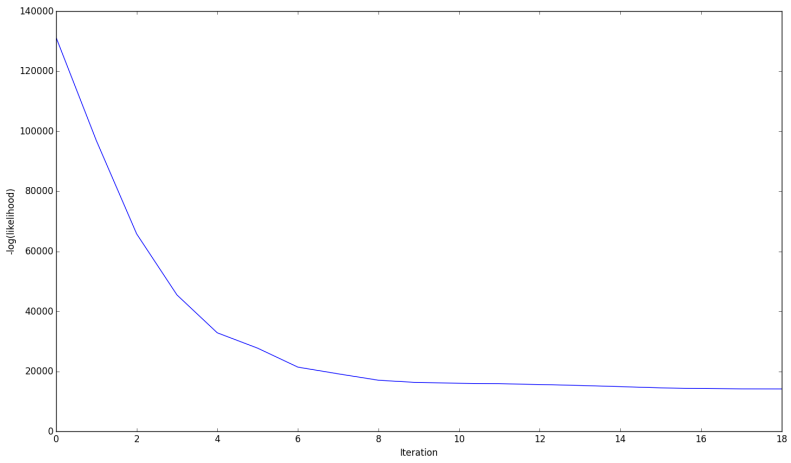
5

Results

5.1 Likelihood

The likelihood converges at about iteration 10 (**Fig 5.1**). The negative log-likelihood on average decreased from trial-to-trial.

Figure 5.1: -log(Likelihood) over 20 iterations.



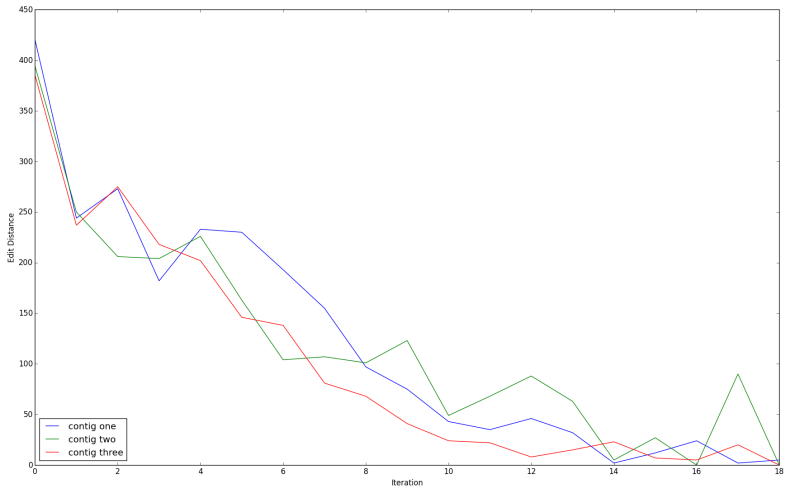
5.2 Evolving Contigs

The edit distance between contig i at time step t and time step t-1 decreases and converges at consensus contigs. (**Fig 5.2**). Edit distances plateaued at about trial 14.

5.3 Read Mapping

k-mers were initially distributed randomly into contigs and offsets (**Fig 5.3**). Our model clustered like k-mers together, merging overlapping contigs and discovering the correct subsequences. (**Fig 5.4**).

Figure 5.2: Percent change in edit distance over 20 iterations.



5.4 Merges

A merge occurred during the displayed trial of the algorithm. In the initialized mapping, there were three contigs of length 1000 (**Fig 5.3**), and in the 20th iteration only two contigs existed, one of length 2000 and the other of length 1000. (**Fig 5.4**).

Figure 5.3: Initial distribution of k-mers at each contig offset.

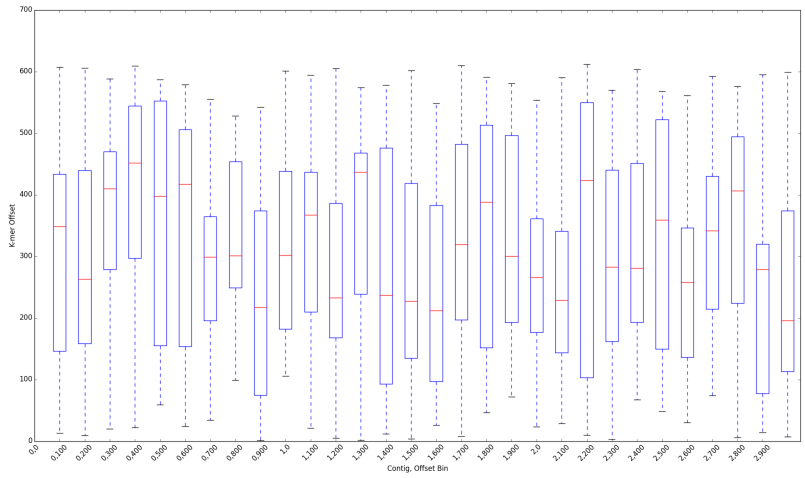
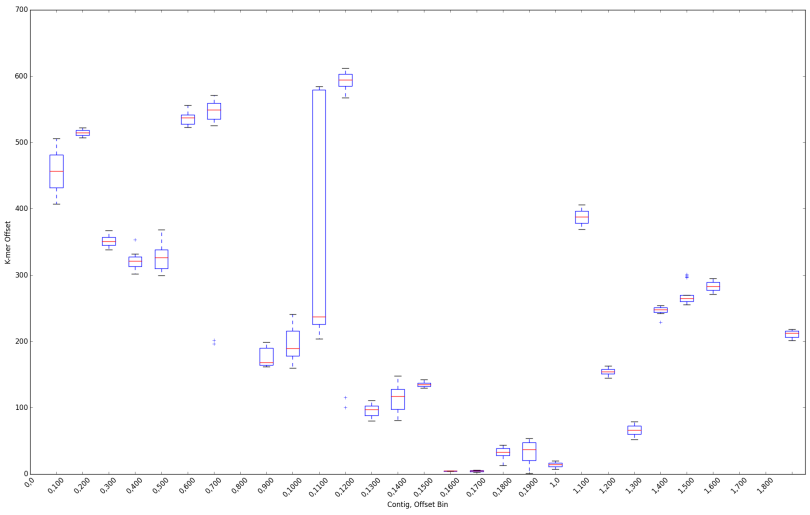


Figure 5.4: Trial 20 distribution of k-mers at each contig offset.



6

Conclusions

6.1 Subsequence Discovery

Our model successfully clusters like k-mers together and is able to identify similar reads and the true subsequences these reads would have been generated from. While we were not able to reconstruct the original sequence, this model can arguably be applied for new species discovery in unknown settings because it is capable of outputting accurate subsequences.

6.2 Likelihood and Convergence

Our model shows statistical properties of correctness because the likelihood function is monotonically decreasing after each iteration and ultimately converges to a single value. Note this likelihood function does not provide any guarantees of reaching a global optimum, but this is okay because the results we have shown give us good reason to believe the convergence has some biological relevance.

6.3 P(O) Distribution and Merging

As we have shown in our results, our model is capable of clustering like k-mers together and merge tails of contigs where we have exact matches. However, once we merge, our algorithm runs into trouble because of limited regularization on the offset random variable, O. This is the precise reason why the current stage of our algorithm has been unable to reconstruct single or multiple sequences from a single run of the model. This regularization would force reads to go to spaces of higher probability, which in our case, this would mean around the "true zero" of the contig where there is higher overlap. Because our regularization on O does not push reads towards spaces of higher overlap, we get reads mapping to offsets far from the "true zero" as you can see in figure 5.4 post-merge.

6.4 Other future work

We would like to implement a prior distribution of $p(o)$ that like $p(s)$ is able reduce the unit length of the variable described. Laserson describes a symmetric-geometric

distribution, centered about $o = 0$ as a possible alternative to our uniform distribution [1]. A possible first distribution could be:

$$p(o) = 0.5(1-p)^{|o|-1}p,\ o \in (-\infty, \infty)$$

Note that this distribution would change offset 0 of the contig to be the center of the contig, and reads would be less likely to be mapped to areas away from the center.

In addition, we would like to attempt to run our data on metagenomic datasets to see whether like reads from different species cluster together. **Fig 5.4** detailed how reads from the same area of the genome were successfully mapped together. Though the contig merge was not completely successful, we still should be able to show that clusters from the same species map together, just showing species differentiation, or possible even horizontal gene transfer if species with like genes cluster about the same area of the contig.

Another challenge that is often faced in metagenomic datasets are that there is low sequence abundance for certain species in a dataset. We experienced low-coverage at the end of the our contigs due to the fact that our reads were every k-mer from a species. This problem would be amplified, and more creative solutions will need to be found.

Lastly, implementing a noise model based upon quality scores would be a more realistic approach to describing the prior, $p(x, y)$. The p_{miss} parameter would be different for each nucleotide, and take into accout the corresponding quality score of that nucleotide for that read.

Bibliography

[1] Laserson, Jonathan et. al. Genovo: De Novo Assembly for Metagenomes. *Journal of Computational Biology. 2011. 18:3 2011.*

[2] Li-Thiao-Te Sebastian et. al. Bayesian model averaging for estimating the number of classes: applications to the total number of species in metagenomics. *Journal of Applied Statistics, 2012 39:7.*

[3] Zagordi, O., Geyrhofer, L., Roth, V., et al. (2009). Deep sequencing of a genetically heterogeneous sample: local haplotype reconstruction and read error correction. *RECOMB 2009 345–358.*

[4] Adler, D. (2016). Homework 4. *Computational Genomics 500.439/639.*

[5] Ray, Priyadip et. al. (2014). Bayesian joint analysis of heterogeneous genomics data. *Bioinformatics, 2014 30: 14*

[6] Langmead, B. (2015). suffixPrefixMatch() *http://nbviewer.ipython.org/github/BenLangmead/com genomics-class/blob/master/projects/UnpairedAsmChallenge.ipynb*

[7] Langmead, B. (2015). Computational Genomics Edit Distance. *http://nbviewer.ipython.org/github/BenLangmead/comp-genomics-class/blob/master/notebooks/CG_DP_EditDist.ipynb*