PHACCS III: A Tool for Predicting Environmental Characteristics of Bacteriophage

Justin W Domes, Bridget K Druken, Peter Salamon

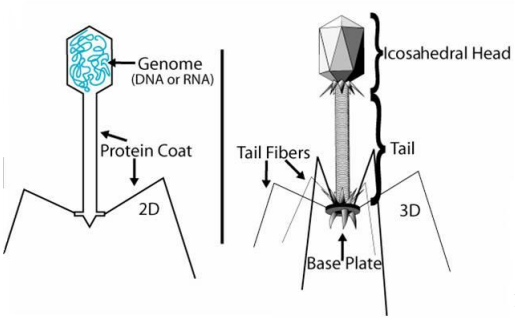
July 27, 2007

Abstract

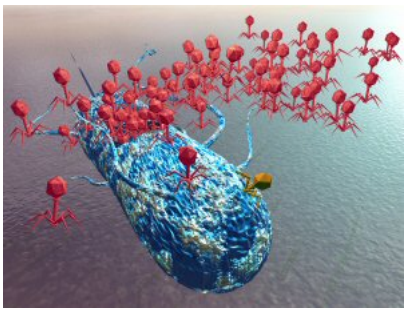
Bacteriophage, bacteria killing viruses, are the most common biological entities in the world and yet very little is known about their abundance and diversity. Such knowledge could have strong implications for the potential reversal of global warming since approximately 25% of the oceans carbon cycles through phage each day. We model phage metagenomic communities using PHACCS III (Phage Communities from Contig Spectra), a program written in MATLAB. Based on combinatorial optimization methods and Markov Chain Monte Carlo simulations, a list of observed contiguous DNA fragments leads to predicted genome lengths and abundances of phage species. Additionally, our program predicts the genomes from which each contig originated.

1 INTRODUCTION

Bacteriophage, commonly referred to as phage, are one of the most common biological entities in the world. Phage have the ability to affect climate, bacterial diversity and genetic exchange, yet very little is known about them. Approximately 25% of the ocean's carbon cycles through phage a day, creating the potential to develop strategies to amiliorate global warming. Phage, sometimes referred to as the new 'magic bullet', have the ability to evolve with bacteria, making a bacterium unable to build resistance to it. This makes phage prime candidates for curing bacterial infections. Also, the ability for phage to self-assemble in a petri dish is attractive to researchers in the field of nanotechnology. Only 510 sequenced genomes of phage exist, with thousands of other existing species of phage yet to be cultured. Since viruses do not have common conserved genetic elements that can be sequenced and used as diversity and evolutionary distance markers, alternative methods are needed in order to gather more metagenomic information of unculturable phage.



[1]


[2]

2 BACKGROUND

Some general background information regarding phage is needed in order to understand the creation of the program PHACCS III.

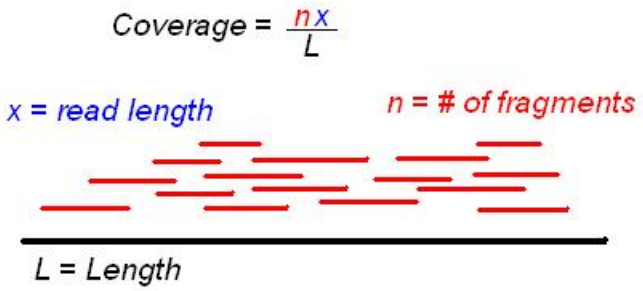
Definitions

A contiguous piece of overlapping DNA strands originating from a single genetic source is referred to as a *contig*. In order for a contig to be created during the assembly process, which will be discussed in the next section, a mimimal overlap of 35 basepairs (bp) must occur with 99% identity. A basepair is a nucleotide within a strand of DNA. A *q-contig* is made up of q fragments where each fragment has an average length of approximately 100 bp. An assembled contig is the resulting strand of DNA sequences created by a q-contig. *Coverage*, which is equal to

$$c = \frac{n \cdot x}{L} \tag{1}$$

[1] Paul Koerbitz "Modeling Phage Populations"
[2] http://www.surrey.ac.uk/SBMS/MicrobialSciences/research/

refers to the average number of fragments in the sample containing any basepair along the genome. The *genome* of an organism or virus holds the hereditary information and its DNA. The number of sequenced genomes of phage are relatively few and are of interest to study because they reveal information about a virus and the static viral community struture.



Assembly Process

A sequencing approach to access information about microbial diversity such as phage is needed for a few reasons. One reason is due to the fact that only a small fraction of environmental microbes are readily cultured [1]. This is due to the fact that each phage species has a relatively small number of possible microbial hosts. Another reason is due to the fact that it is still difficult to learn about phage under an electron microscope. Hence it is necessary to gain access to environmental phage diversity through means of shotgun sequencing.

The process of shotgun sequencing during which contigs are assembled involves a few steps. In the present case, the sample data was taken from the Bay of British Columbia. The process involves collecting gallons of seawater from the environment in order to filter out bacteria, eukaryotes and large particles. The filtering techniques involved include using a combination of differential filtration and gradient centrifugation in order to isolate phage [2]. DNA is then extracted, untwisted and broken into 100 bp fragments so that DNA sequences can be read. Each fragment has a certain length and abundance in the environment. Contigs are then assembled using a sequencing program. The information yielded after assembly is a list containing the length of contig measured in basepairs and the number of fragments that make up the contig, among other information.

PHACCS - PHAge Communities from Contig Spectra

PHACCS is a free online tool devloped by researchers at San Diego State Univeristy (SDSU). The broad purpose of the PHACCS program is to estimate the structure and diversity of uncultured viral communities using metagenomic information [1]. Utilizing a modified Lander–Waterman algorithm, the program creates a virtual contig spectrum with which to compare the user–inputed experimental contig spectrum. The program works by optimizing the model parameters until the the two aforementioned spectra are as closely matched as possible. Once this optimal state has been reached the user is given output with regards to the phage community evenness, richness and diveristy, as well as the rank–abundance of the various genotypes.

Due to advances in biological techniques that are used to sequence DNA, the original PHACCS does not perform well when dealing with large data sets. Since the assembly process is more efficient, less data is lost during the filtration period which results in larger contigs. PHACCS also uses an average length of 50,000 bp for a genome, when it is known that genome lengths vary from 2,000 bp to over 250,000 bp. More data is used in PHACCS III in order to model sample community characteristics such as rank, abundance and now length of the genome.

The figure below shows a distribution of genome lengths of the 510 sequenced phage genomes in GenBank. The graph gives good reason for allowing genome lengths to vary as the program iterates.

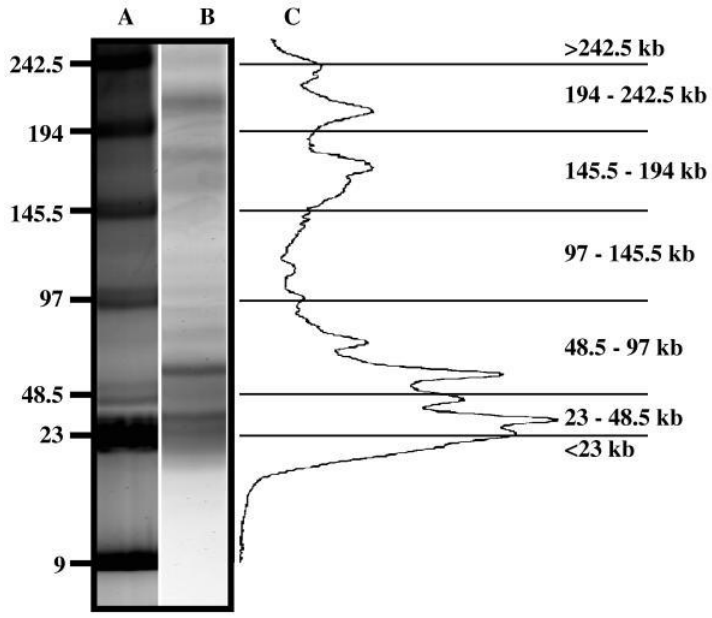


Figure 1: Distribution of Sequenced Genome Lengths from GenBank

3 THEORY

Markov Chain Monte Carlo

The idea of simulation using a Markov Chain Monte Carlo (MCMC) method is one of practical importance in Bayesian statistics and computational physics. The purpose of this type of algorithm is to select from a probability distribution such that a Markov Chain is constructed with the desired distribution as the stationary distribution. We use it here as a means to produce a distribution of optimal states from which inferences can be made about the static properties of the phage environmental structure. As stated in the previous section, while the program travels across the likelihood landscape according to the Metropolis-Hastings approach, a Markov Chain of contig states is generated. The program acquires its label as a Monte Carlo simulator due to the fact that each candidate state is chosen from a uniform distribution using a random number generator in MatLab.

Metropolis Hastings

Our technique for likelihood optimization is a variation on the versatile Metropolis-Hastings algorithm. This method involves sampling from a probability distribution, in our case a distribution of the likelihood function over all possible configurations of contigs within genomes, comparing the sample, called the candidate state, to the likelihood of the current state, and accepting the candidate state as the next state with a probability equal to the ratio of the candidate state likelihood to the current state likelihood. We see that if we have the current state likelihood,

$$\theta_0 = L(x^t)$$

and the candidate likelihood

$$\theta = L(x^{t+1})$$

then our ratio is,

$$\alpha = \left(\frac{\theta}{\theta_0}\right)^k$$

where k = 1,2,3,... In accordance to the true Metropolis-Hastings approach one would then make the move with probability $min(\alpha, 1)$. However, in our program we assume that $k = \infty$, and effectually allow no drop in the likelihood as we continue to progress to a maximum likelihood state. After an arbitrary number of moves are made, remembering that no drops in likelihood are allowed, we reach a state from which no significant increase in the likelihood will occur, or a local maximum of the likelihood function. We call this state optimal. Also, because the Metropolis-Hastings process results in the creation of a Markov Chain, we may choose to refer to it in the language of Markov Chains as the steady state.

Likelihood Function

In contrast to PHACCS I which was ineffective when dealing with high coverage data, the likelihood function optimized in PHACCS III was developed under the assumption that the

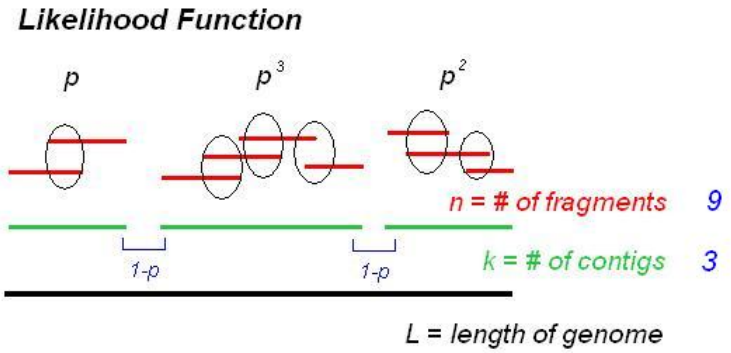


Figure 2: The likelihood of k contigs forming a genome of length L composed of a total of n fragments

length of a genome of a certain phage virus is the sum of all contigs k within that genome. Hence the lengths of unsequenced genomes can be estimated with only the knowledge of the contigs.

The likelihood function in PHACCS III finds the probability that a certain number of contigs, k, assemble together to form a genome of length L, where the genome is indexed by j. Within each sample environment, contigs are used to assemble M different genomes, where M is the assumed total number of genomes in the sample, and are treated as independent events. Hence, the likelihood of k contigs forming M genomes is

$$\text{Likelihood} = \prod_{j=1}^{M} k_j!(1-p_j)^{k_j-1} p_j^{n_j-k_j} \tag{2}$$

where $p = 1 - e^{-nx/L}$ is probability of overlap, $k = \#$ of contigs in genome, $n = \#$ of fragments in genome, and $j =$ index of genome.

The $k!$ combinatorial term counts the number of ways k contigs can be assembled to form the genome. The p term is the probability that an overlap of fragments occured to form the contig. The number of times p occurs is the total number of fragments making up the genome, n, minus the number of contigs in the genome, k. The probability of a gap occuring between contigs is $(1-p)$ and occurs $k-1$ times. This term is necessary to consider since excluding it would then assume the existence of one large contig covering a genome.

Another term of interest added into the likelihood function deals with the distribution of lengths given by a certain genome coverage[3]. The distribution assigns a probability that a contig with a certain length belongs to a genome of a certain coverage. The term is added into the likelihood function under the assumption of independence.

$$\text{Likelihood} = \prod_{j=1}^{M} k_j!(1-p_j)^{k_j-1} p_j^{n_j-k_j} \cdot \text{KorFactor} \tag{3}$$

4 PROGRAM

PHACCS III - PHAge Communities from Contig Spectra

PHACCS III, a computer program created in MATLAB, is the name of the MCMC simulator created to deal with the problems encountered by the original PHACCS program as discussed in Section 2. As input, PHACCS III takes a list of observed contigs, their lengths, and whether or not each contig had a BLAST hit to any of the 510 fully-sequenced genomes found in GenBank. BLAST is a program that aligns genomes based on a likelihood function. Using combinatorial optimization methods, a distribution of optimal states is reached. The optimal states signify the most likely arrangement of contigs into specific genomes and also predicts the lengths, coverages, and, subsequently, the abundances of genomes.

Here is an example in Table 1 of an input into PHACCS III that comes from an environment with assumption of two species.

PHACCS III Input Matrix	
Length of Contig	Number of Fragments
972	80
219	21
5521	487
1210	105
2391	216
1608	136
252	9
262	9
292	17
75	3
⋮	⋮

Table 1: **Example of an input into PHACCS III from a sample environment with assumption of two species**

The program begins by randomly assigning a genome to each contig. The order of the assignment of the indices of the genomes is called a state (shown in Table 2 the first column in magenta). We can obtain the coverage of each contig easily by Equation (1), resulting in

Start State Arrangement			
Genome Index	Length of Contig	Number of Fragments	Coverage
1	972	80	5.3
2	219	21	6.2
1	5521	487	5.7
2	1210	105	5.6
1	2391	216	5.9
1	1608	136	5.5
2	252	9	2.3
1	262	9	2.2
2	292	17	3.8
2	75	3	2.6
⋮	⋮	⋮	⋮

Table 2: **Example of a start state along with each contig's length, number of fragments and coverage**

The first column in Table 2 is an example of a current state because the entries in the columns corresponding to Contig Length, Number of Fragments and Coverage remain unchanged as genome indices are reassigned. The next step involves randomly choosing a row from the start state, the first row say, and changing the index of its genome, creating the candidate state in Table 3:

Candidate State Arrangement			
Genome Index	Length of Contig	Number of Fragments	Coverage
2	972	80	5.3
2	219	21	6.2
1	5521	487	5.7
2	1210	105	5.6
1	2391	216	5.9
1	1608	136	5.5
2	252	9	2.3
1	262	9	2.2
2	292	17	3.8
2	75	3	2.6
⋮	⋮	⋮	⋮

Table 3: **Example of a candidate state where the genome of the first row is changed from a 1 to a 2. The table also contains each contig's length, number of fragments and coverage**

The ratio of the likelihood of each state is compared.

$$\text{Ratio} = \frac{Likelihood(CandidateState)}{Likelihood(CurrentState)}$$

If the ratio is greater than one, then the Candidate State is accepted and becomes the new Current State. If the ratio is less than one, then the move is not made. In both cases, another contig is randomly selected and the process repeated. After an arbitrary number of moves, a steady state is reached after which there will be no increase in the likelihood. This is called the optimal state. After this entire procedure is conducted a sufficient number of times, we obtain a distribution of only the optimal states, from which inferences can be made about the true lengths, abundances and arrangement of contigs into genomes. A schematic of the process is shown Figure 3.

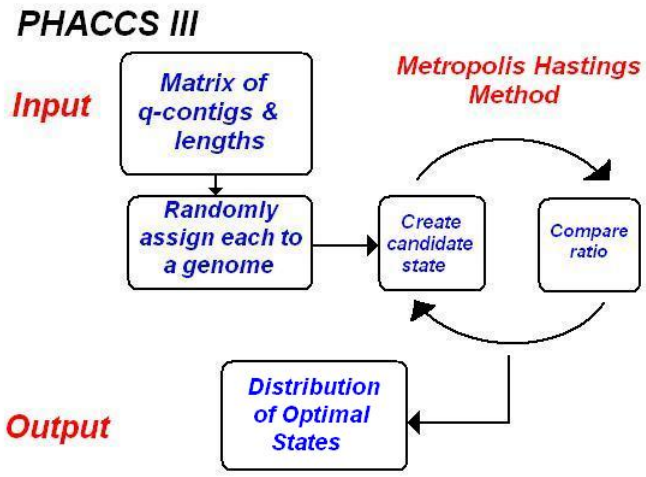


Figure 3: **Flow-chart of PHACCS III algorithm**

RESULTS

The accomplishments of PHACCS III range from successful assembly of virtual contig data to correctly assigning observed BBC data into genomes. One of the first goals was to simulate contigs and their lengths in order to follow the accuracy of the output of the likelihood function. This was accomplished through the GenerateSampleContigs function.

A recent addition to the likelihood function is the GenBank hits reward that is given to a contig when assigned to a genome of approximately the same length as a genome to which the contig had a BLAST hit. The lengths of the genome to which the contig hit are then categorized into one of the seven length intervals created from the actual lengths of known phage genomes from GenBank. Each entry in the seven interval vector is normalized so that a probability vector is created. If the length of the contig to which the genome is currently assigned falls into one of the non-zero length intervals of the probability vector, the log of that probability is added to the likelihood function.

More additions to the likelihood function are needed so that the true state generates the maximum likelihood of all possible states.

References

[1] F. Angly, B. Rodriguez-Brito, D. Bangor, P. McNairnie, M. Breitbart, P. Salamon, B. Felts, J. Nulton, J. Mahaffy, and F. Rohwer. *PHACCS, an online tool for estimating the structure and diversity of uncultured viral communities using metagenomic information.* 2005.

[2] M. Breitbart, P. Salamon, B. Andresen, J. Mahaffy, A. Segall, D. Mead, F. Azam, and F. Rohwer. *Genomic analysis of uncultured marine viral communities.* 2002.

[3] Paul Koerbitz. *Modeling of Phage Communities.* 2006.

APPENDIX

```
% Main file that holds the information from which a virtual list of contigs
% is generated

clear
fragLength = 65;

% Number of reads (=fragments)
N=1000;

% Length of f & L must equal # of viral species
% Relative abundances
f=[1/2,1/3,1/6];
% Lengths of genomes of species
L=[500,5000,15000];

% Data = [true genome index,length,q,coverage] as a row for each contig
Data=GenerateSampleContigs(N,f,L,fragLength);

-----

function Data=GenerateSampleContigs(N,f,L,fraglength)
% GenerateSampleContigs : Function which generates virtual contigs.

% variables:
% N = number of fragments
% f = vector of relative abundances of the species,
% L = vector of length of each species genome,
% fraglength = effective average fragment length

% Randomly assigns fragments to each genome of viral species
genomes=rand_from_probvector(f.*L/sum(f.*L),N);
genomes=sort(genomes);
contigsNew = [];
genomeIndex = [];
for i = 1:length(L)
    % Gives number of fragments of genome i
    Ni = sum(genomes==i);
    NVec(1,i) = Ni;

    % Generates virtual contigs & lengths for one species, stores them in
```

```
    % vector
    contigHolder = FindContigs(NVec(1,i),L(i));

    % Holds contigs for all species
    contigsNew = [contigsNew;contigHolder];

    % Creates genome index for each contig
    genomeIndex = [genomeIndex;i*ones(size(contigHolder,1),1)];
end

real_coverages=fraglength*Ni./L;
coverages=fraglength*contigsNew(:,2)./contigsNew(:,1);

% Vector that stores whether or not a contig virtually hit GenBank
genBankHit = zeros(size(genomeIndex,1),1);

for i = 1:size(genomeIndex,1)
    genbankRand = rand;

    % 20% of contigs in the top 100 high coverage contigs from BBC data
    % hit GenBank
    if genbankRand < 0.2
        % If contig i hit GenBank, length of genome from which contig came
        % is known
        genBankHit(i,1) = L(genomeIndex(i,1));
    end
end

-----

function contigs=FindContigs(N,L)
% FindContigs: Function which takes in number of fragments and length of
% genome and outputs length of contig and number of fragments [l,q]

% variables:
% N = number of fragments
% L = length of genome

if N==0
    contigs=[]
    return
end
```

```
% Assumes actual average fragment length of 100 bp with minimul overlap of
% 35 bp
FragLength=65;

% Sorts random starting positions of fragments within a genome
positions=sort(ceil(rand(N,1)*L));
augmented_positions=[0;positions;L];

% Can be used to visually see fragments overlapping to form contigs
% for i=1:N
%     plot([positions(i),positions(i)+65],[i,i])
%     hold on
% end
% hold off

% Finds the distance between starting positions of fragments
spacing=diff(positions)

% Logical vector that finds spacing greater than 65 bp
gaps=spacing>65

% Yields the number of fragments overlapping up to the gap
breaks=find(gaps)
augmented_breaks=[0; breaks; N];
q=diff(augmented_breaks);
numcontigs=length(q);

% If no gaps, then there is only one contig
if length(breaks)==0
    contigs=[positions(N)-positions(1)+65,N];

else
    lengths(1)=positions(breaks(1))-positions(1);

    % Finds lengths of contigs
    for i=2:numcontigs
        lengths(i)=positions(augmented_breaks(i+1))-positions(augmented_breaks(i)+1);
    end
    contigs=[lengths'+65,q];
end

-----
```

```
% Main file for PHACCS III

% Number of contigs in our sample
Nstate=size(Data,1)

% Number of species from which the contigs came
Nspecies=max(Data(:,1))

% Fix the first entry for easy reading
startstate(1,1)=Nspecies;

% Randomly assigns each contig to one of the Nspecies
startstate(2:Nstate,1)=unidrnd(Nspecies,Nstate-1,1)

% Calculates the log likelihood of the start state
currentLogL=TotalLogLikelihood(startstate,Data);

% Stores the "correct" configuration of contigs within the genomes
Truestate=Data(:,1);

% Calculates the log likelihood of the "correct" state
TrueLogL=TotalLogLikelihood(Truestate,Data)

% Sets the current state as the start state
currentstate=startstate;
optimalstates=[];
optimalLogLikelihoods=[];

% Number of optimal states desired
for trial=1:5
    % Variable that indicates when the optimal state has been reached
    tired=false;

    % Initializes tiredness counter, number of times program should perform
    % quasi Metropolis-Hastings method.
    tirednesscounter=0;

    % Randomly assigns genome index to each contig
    currentstate(2:Nstate,1)=unidrnd(Nspecies,Nstate-1,1);

    % Log likelihood of current state
    currentLogL=TotalLogLikelihood(currentstate,Data);
```

```
    while not(tired)
        % Creates candidate state
        trialstate=jiggle(currentstate);
        % Log likelihood of candidate state
        trialLogL=TotalLogLikelihood(trialstate,Data);

        % Quasi Metropolis-Hastings method: check to see if ratio is
        % greater than 1. If yes, candidate state becomes current state.
        % Tiredness counter resets to 0.
        if trialLogL>currentLogL
            currentstate=trialstate;
            currentLogL=trialLogL;
            tirednesscounter=0;
        end

        tirednesscounter=tirednesscounter+1;

        % After an arbitrary number of trials where likelihood has not
        % improved, 1000 in our case, optimal state is reached
        if tirednesscounter>1000
            tired=true;
            optimalstates=[optimalstates,currentstate];
            currentLogL=TotalLogLikelihood(currentstate,Data);
            optimalLogLikelihoods=[optimalLogLikelihoods,currentLogL];
        end
    end
end

% Resulting range of optimal states and their likelihoods
optimalstates
optimalLogLikelihoods

% "Correct" likelihood (used for virtually-generated data)
TrueLogL=TotalLogLikelihood(Truestate,Data)

-----

function y=TotalLogLikelihood(state,Data)
% TotalLogLikelihood : Function that takes a current state of contigs with
% a genome index and data, virtual or observed, and outputs the likelihood

% variables:
% state = current state of contigs with certain genome indices
```

```
% Data = virtual or observed data, in the form of
% [genomeIndex,contigsNew,coverages,contigIndex,genBankHit]

fraglength=65;

% Number of species
Nspecies=max(state);

index=zeros(Nspecies,size(Data,1));

% Initialize y, the log likelihood, to be 0
y=0;

% Length, L, of genome i is determined by summing up all contigs indexed i
% since high coverage is assumed
for i=1:Nspecies
    indexi= state==i;
    % k = number of contigs that make up genome i
    k=sum(indexi);

    % n= sum of all fragments that make up genome i
    n=sum(Data(indexi,3));

    % L = length of genome i
    L=max([1,sum(Data(indexi,2))]);

    % Kor Matrix consists of length of contig, # of fragments that compose the
    % contigs, index of contig, and GenBank hit
    KorMatrix=Data(indexi,[2,3,5,6]);

    % Koerbitz factor for one genome
    KoerbitzTerm=KorProd(KorMatrix,n,L,fraglength);

    if k*n>0
        y=y+logLikeKbtz(KoerbitzTerm,k,n,L,fraglength);
    end
end

-----

function y=KorProd(KorMatrix,n,L,fraglength)
% KorProd: Function that inputs takes into account the probability of a
% contig of a certain length being a part of a genome with certain coverage
```

```
% and yields a log probability of the configuration of the contigs.

% variables:
% Kor Matrix = m = # of fragments that make up the contigs
%              l = lengths of contig
%              fraglength = effective average fragment length
% n = # of fragments that make up the genome
% L = length of genome
% fraglength = effective average fragment length

KorFactorProduct=0;

% Quasi coverage: number of fragments, n, divided by length of genome, L
alpha = n/L;

for i=1:size(KorMatrix,1)

    % # of fragments belonging to contig i
    q=KorMatrix(i,2);

    % length of contig i
    contigLength=KorMatrix(i,1);

    % index of contig i
    contigIndex = KorMatrix(i,3);

    % GenBank hit of contig i
    GBHit = KorMatrix(i,4);

    if GBHit ~= 0

        % Virtual GBHit
        if GBHit ~= 1
            LUpBd = L + 3000;
            LLowBd = L - 3000;
            % Reward the likelihood of the current state if in a certain
            % range
            if ((GBHit > LLowBd)&&(GBHit < LUpBd))
                koerbitzReward = 10;
            end
        else
        % Observed data GenBank hit
```

```
        % koerbitzRewardVec is a vector which has 7 ranges of lengths of
        % genomes: 0-5000, 5000-20000, 20000-40000, 40000-60000,
        % 60000-125000, 125000-200000, 200000-250000

        % Generates vector with tallies of GenBankHit genome lengths in
        % proper range
        koerbitzRewardVec = GenBankHitData(contigIndex);

        % Makes koerbitzRewardVec a probability vector
        koerbitzRewardVec = koerbitzRewardVec*(1/sum(koerbitzRewardVec));

        % Finds range of L, which is the length of the genome to which the
        % contig currently belongs
        genomeRangeIndicator = lengthCounter(L);

        % Returns corresponding probability reward contig has for being part
        % of a certain length genome
        koerbitzRewardVec = koerbitzRewardVec.*genomeRangeIndicator;

        % Isolates the probability from zero entries of Koerbitz reward
        koerbitzReward = koerbitzRewardVec(find(koerbitzRewardVec));

        % If contig belongs to a genome of different length than GenBank
        % hit, then no reward given
        if isempty(koerbitzReward);
            koerbitzReward = 0;
        else

        koerbitzReward = log(koerbitzReward);

        end
        % Adds koerbitzReward to likelihood function
        KorFactorProduct = KorFactorProduct
        + log(negbin_kor(q,contigLength,alpha,fraglength))+ koerbitzReward;

        end
    else
        %'no hit'
        KorFactorProduct = KorFactorProduct
        +log(negbin_kor(q,contigLength,alpha,fraglength));

    end
end
```

```
y=KorFactorProduct;

-----

function Y=negbin_kor(q,k,alpha,xtilda)
% negbin_kor: Function that assigns to each contig a probability of being
% within a genome with a certain alpha value (quasi-coverage).

% variables:
% q = # of fragments that compose the contig in question
% k = length of the contig in question
% alpha = m/L  ;where m is # of fragments within the genome, and L the
%               length of the genome to which the contig in question currently
%               belongs
% xtilda = the average effective fragment length

% Paul Koerbitz's expected value and variance from the distribution as
% presented within his masters thesis
mu = (1-(1-alpha)^xtilda*(alpha*xtilda+1))/(alpha*(1-(1-alpha)^xtilda));
varkor = (1-alpha)*
(1+(1-alpha)^xtilda*(2*(alpha-1)+(1-alpha)^(xtilda+1)-alpha^2*xtilda^2))/
(alpha^2*(1-(1-alpha)^xtilda)^2);


if q==1 %if a one contig
    muecontig=mu;
    varecontig=varkor;

    p = muecontig/varecontig;
    r = muecontig^2/(varecontig-muecontig);

    Y = exp((gammaln(k+r)-gammaln(k)-gammaln(r))*p^r*(1-p)^k);
    return
end

% Binomial estimator of the above distribution (also presented in Koerbitz's
% masters thesis)
muecontig = (q-1)*mu;
varecontig = (q-1)*varkor;

p = muecontig/varecontig;
r = muecontig^2/(varecontig-muecontig);
```

```
Y = exp((gammaln(k+r)-gammaln(k)-gammaln(r))*p^r*(1-p)^k);
%keyboard

-----

function y=logLikeKbtz(KorFactor,k,n,L,fraglength)
%logLikeKbtz: Function which determines the loglikelihood of a state

% variables:
% n = # of frags in genome
% k = # of contigs in genome
% q = # of frag within the jiggled contig
% contigLength = length of the jiggled contig
% L = length of the genome (considered to be total length of all contigs due
% to high coverage of the genome)

% Probability of two fragments overlapping
p = 1 - exp(-n*fraglength/L);

% Approximation of log(k!)
logkfact=k*log(k)-k+log(k*(1+4*k*(1+2*k)))/6+log(pi)/2;

% Loglikelihood
y=(logkfact)+KorFactor+(k-1)*(-n*fraglength/L)+(n-k)*log(p);

-----

function GenBankProb = GenBankHitData(contigIndex)
% GenBankHitData: Function which calculates GenBank hits within the ranges
% specified by lengthCounter

% GenBankHitMatrix is a matrix of the contigs within the top 100 highest
% coverage contigs from Bay of British Columbia data. First column is the
% contig index, with the following columns containing the lengths of the
% genomes in GenBank with which the contig had a blast hit.

GenBankHitMatrix=[5158 43769 39898 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
4819 56425 56425 56425 56425 55986 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

0 0 0 0 0 0
28962 42415 42415 34542 39043 39325 110865 63239 39233 39233 4594 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
12228 37357 37357 32316 53373 34371 45789 26607 49220 49220 49220 156102 156102
51141 49826 45503 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
12246 37639 75931 36717 36717 36717 252401 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
39193 36466 36466 47538 35466 35466 35466 49223 47675 42619 44082 41401 43155 45286
44283 43883 43071 40582 40582 43681 52797 156102 156102 156102 41774 41796 41796
47399 67480 67480 50913 50913 42519 42519 196280 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
13588 62787 62787 62787 62706 62706 62706 62706 59866 59866 59866 59866 59866 61765
61765 61765 61765 57416 57416 57416 61670 61670 61670 60942 60942 60942 50559 50559
50559 39505 39505 39505 63882 63882 63882 63882 63882 65195 65195 39043 39043 38297
38297 42638 42663 42493 63239 63239 63239 36107 68999 68999 6760 69777 69777 37357
37357 37357 37357 37357 244835 56902 56902 56902 58638 58638 70797 37359 35580 35580
57050 57050 44373 44373 40331
13180 49940 49940 40794 40794 104820 104820 104820 104820 17904 17904 13944 13944
42460 42460 41308 41308 6068 6068 50913 49575 49575 47537 47537 42663 42663 42663
42638 42638 42638 42493 42493 42493 50550 41318 58128 58128 58128 58128 45923 44777
43095 44970 42415 42415 40014 68999 68999 68999 68999 68999 68999 67480 67480 67480
49220 52297 156102 156102 41774 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
29178 43769 44818 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
21858 44427 44427 57416 63882 18600 75931 75931 57416 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
32347 46012 41409 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
21433 244835 156102 156102 52797 52797 44427 44427 65195 65195 58128 46339 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
53085 252401 252401 129908 129908 196280 57050 57050 178249 178249 23089 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
6848 68999 68999 42465 42465 42465 75931 65195 65195 65195 57050 57050 39898 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```
9212 164018 164018 178249 196280 252401 19033 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
26747 58128 30889 30889 63649 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0
6243 164018 41318 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
14372 63239 63239 63239 49220 49220 49220 49220 37639 42415 42415 57050 57050 57050
47399 252401 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
13925,40794,40794,52797,52797,52797,46012,58128,58128,18600,18600,18600,18600,156102,156
49803 42638 42638 42638 42638 42663 42663 42663 42663 42663 42663 42663 42663 42663
42663 42663 42663 42663 42663 42663 42663 42663 42663 42663 42663 42663 42663 42663
42663 42663 42663 42493 42493 42493 42493 42493 42493 42493 42493 42493 42493 42493
42493 42493 42493 67480 67480 58498 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0];

GenBankProb = zeros(1,7);

% Exclude first column
hitIndex = [2:size(GenBankHitMatrix,2)];

% Finds the row in GenBankHitMatrix of the contigIndex
checkIndex = GenBankHitMatrix(:,1) == contigIndex;

% Puts each length in checkIndex row into a range as specified by
% lengthCounter
GenBankProb(1,:)=lengthCounter(GenBankHitMatrix(checkIndex,hitIndex));

-----

function CountVector=lengthCounter(vector)
% lengthCounter: Function which sorts lengths of GenBank hits into 7
% categories

% variables:
% vector = row in GenBankHitMatrix with lengths of genomes to which the
% contig had a blast hit

% 7 different ranges
categories=[0,5000
```

```
            5001,20000
            20001,40000
            40001,60000
            60001,125000
            125001,200000
            200001,250000];

CountVector=zeros(1,7);

% Cycles through the length of vector and classifies each length into
% length interval
for i=1:length(vector)
    if vector(i) == 0
        break
    end
    CountVector(category(vector(i))) = CountVector(category(vector(i))) + 1;
end

-----

function cat=category(lengthOfGenomeHit)
% category: Function which takes a length of a genome and sorts it into 1
% of 7 ranges

% varibles:
% lengthOfGenomeHit = length of genome to be categorized

% 7 different ranges
categories=[0,5000
            5001,20000
            20001,40000
            40001,60000
            60001,125000
            125001,200000
            200001,250000];

classified = false;

% Counter initializes as 7
counter = size(categories,1);

% Sort lengthOfGenomeHit into proper length range
while(~classified)
```

```
    if lengthOfGenomeHit > categories(counter,1)
        cat = counter;
        classified = true;
    else
        counter = counter - 1;
    end
end

-----

function jiggledstate=jiggle(currentstate)
% jiggle: Function which takes the current state and switches the species
% of a contig.

% variables:
% currentstate = current state of assigned genome indices to contigs

n=length(currentstate);

% set the first contig to be the max number of species
Nspecies=currentstate(1);

% randomly selects an index of a contig, doesn't jiggle first coordinate
jiggledindex=unidrnd(n-1,1,1)+1;

jiggledstate=currentstate;

% force the selected contig to change to a species index other than
% the current
jiggledstate(jiggledindex)=mod(currentstate(jiggledindex)
+(unidrnd(Nspecies-1,1,1)-1),Nspecies)+1;

-----

function uni=unidrnd(M,n,m)
% unidrnd: Function that returns nxm matrix of uniform random integers 1 to M

uni=ceil(rand(n,m)*M);

-----

function r=rand_from_probvector(pvec, n,m)
% rand_from_probvector: Function that takes into account relative abundances
```

```
% and generates virtual fragments for each species.

% variables:
% pvec = relative abundances of species
% n = number of fragments
% m = number of species

% Corrective element if number of arguments input is only 1 or 2
if nargin<3
    m=1;
end
if nargin<2
    n=1;
end

% Number of species
ncases=length(pvec);

% Partitions set [0,1] based on relative abundances of species
tester=cumsum(pvec);

% Generates an nxm random matrix
r=rand(n,m);

% Uses the random number matrix to assign each fragment to one of the
% species
for i=1:ncases
    r(r<tester(i))=i;
end

-----

% Main file that holds the data from Bay of British Columbia

% 100 highest coverage contigs from the BBC sample.
% Column 1: contig index
% Column 2: contig length
% Column 3: # of fragments that compose the contig
% Column 4: GenBank hit logical
% Column 5: Contig coverage

BBCDataMatrix = [37934 1199 5248 0 284.5037531
6602 426 1307 0 199.4248826
```

```
25294 612 1563 0 166.004902
16218 714 1110 0 101.0504202
10050 445 470 0 68.65168539
53392 1122 614 0 35.57040998
5153 305 140 0 29.83606557
10298 404 152 0 24.45544554
13232 528 183 0 22.52840909
48263 200 62 0 20.15
10199 384 118 0 19.97395833
20232 1421 415 0 18.98311049
49956 1934 564 0 18.95553257
49911 492 126 0 16.64634146
6091 451 113 0 16.28603104
41856 189 47 0 16.16402116
46915 376 91 0 15.73138298
49803 368 88 1 15.54347826
40163 158 36 0 14.81012658
28505 892 202 0 14.71973094
33419 175 39 0 14.48571429
5158 1309 291 1 14.4499618
4819 1898 412 1 14.10958904
1570 108 23 0 13.84259259
28962 1401 294 1 13.64025696
9498 751 157 0 13.5885486
17914 480 93 0 12.59375
53287 383 74 0 12.55874674
16545 99 19 0 12.47474747
21779 327 62 0 12.32415902
46647 655 122 0 12.10687023
46855 3481 646 0 12.06262568
10276 267 48 0 11.68539326
48572 400 71 0 11.5375
38462 361 64 0 11.52354571
37846 1780 310 0 11.32022472
50989 1514 256 0 10.99075297
40110 1474 249 0 10.98032564
52946 190 32 0 10.94736842
54143 595 99 0 10.81512605
10756 828 136 0 10.6763285
12246 1354 222 1 10.65731167
35761 545 88 0 10.49541284
12228 448 72 1 10.44642857
31120 490 78 0 10.34693878
```

```
22540 2318 359 0 10.06686799
20594 488 75 0 9.989754098
43894 596 91 0 9.924496644
48830 171 26 0 9.883040936
39193 2503 370 1 9.608469836
13588 1208 176 1 9.470198675
18936 412 60 0 9.466019417
16644 383 55 0 9.334203655
46490 676 97 0 9.326923077
26359 379 54 0 9.26121372
13180 1582 225 1 9.244627054
29178 394 56 1 9.23857868
32840 549 78 0 9.234972678
28253 388 55 0 9.213917526
21858 878 122 1 9.031890661
32347 772 107 1 9.009067358
7929 1025 141 0 8.941463415
41365 371 51 0 8.935309973
27909 1683 228 0 8.8057041
43983 748 101 0 8.776737968
28187 824 111 0 8.756067961
42309 744 100 0 8.73655914
25338 740 99 0 8.695945946
3408 369 49 0 8.631436314
48394 381 50 0 8.530183727
21433 1079 141 1 8.493975904
13507 1496 194 0 8.429144385
18083 1180 153 0 8.427966102
4528 247 32 0 8.421052632
26678 1509 195 0 8.399602386
27791 488 63 0 8.391393443
41985 527 68 0 8.387096774
16106 419 54 0 8.377088305
50070 2028 261 0 8.365384615
39081 884 113 0 8.308823529
21261 431 55 0 8.294663573
24111 110 14 0 8.272727273
54089 150 19 0 8.233333333
20130 595 75 0 8.193277311
28212 735 92 0 8.136054422
17844 1154 143 0 8.054592721
24351 317 39 0 7.996845426
43103 840 103 0 7.970238095
```

```
53085 1162 142 1 7.943201377
6848 649 79 1 7.912172573
33672 282 34 0 7.836879433
9212 707 85 1 7.814710042
21495 226 27 0 7.765486726
15249 153 18 0 7.647058824
12587 485 57 0 7.639175258
13925 928 109 1 7.634698276
40201 478 56 0 7.615062762
26747 3921 458 1 7.592450905
6243 1290 150 1 7.558139535
14372 1351 157 1 7.553663953];

bbcIndex = BBCDataMatrix(:,1);
bbcContigLength = BBCDataMatrix(:,2);
bbcContigFrags = BBCDataMatrix(:,3);
bbcGBHit = BBCDataMatrix(:,4);
bbcCoverage = BBCDataMatrix(:,5);

% Randomly assigns each contig to a genome as a start state.
Answer=unidrnd(9,size(BBCDataMatrix,1),1);
Data=[Answer,bbcContigLength,bbcContigFrags,bbcCoverage,bbcIndex,bbcGBHit];
GuessContigs
```