Metagenomic Assembly using Coverage and Evolutionary distance (MACE)

Introduction

Recent advances in accuracy and cost of whole-genome shotgun sequencing have opened up various fields of study for biologists. One such field is that of metagenomics – the study of the bacterial biome in a given sample. The human body, for example, has approximately 10 bacterial cells for every native cell\cite{1}, and lack of certain strains may contribute to prevalence of diseases such as obesity, Crohn's disease, and type 2 diabetes\cite{2,3,4}. Although the field of metagenomics is very promising, as with any new field, there are many difficulties with such analyses.

Arguably, the greatest obstacle to sequencing a sample’s biome is the existence of a large number of strains of bacteria, many of which do not have an assembled genome, and an unknown prevalence of each strain. Assembly is further complicated by inherent inaccuracy in reading DNA fragments, which makes it difficult to distinguish between noise and true variants either within a strain or between strains. Additionally, in a given biotic sample, there are bound to be many clusters of closely related strains. When related bacteria coexist in a sample, any conserved regions can easily cause misassembly of the genomes. For example, two genomic sequences with a single region of homology (of length greater than the read length) will be easily distinguished from each other both before and after the homologous region. However, if given only short reads from the two genomes, it becomes impossible just based on the sequences to infer which sequence before the homologous region belongs to which sequence after the homologous region (Figure 2). It becomes a non-trivial problem to assign assembled sequences before a homologous region to ones after it, and this will be the problem this paper addresses.

Background

Assembly is a type of superstring reconstruction problem. The idea of it is as follows: given any number of strings, or sequences of letters, generate the shortest string (called the superstring) that has every given string as a substring. In the context of genomics, it translates to the following: given a large set of short reads generated from a genome, assemble the original genome. Before 2001, virtually all efforts to assemble genomes employed an “overlap-layout-consensus” method. This is a very intuitive, yet ultimately very difficult way to solve the problem of assembly. It involves finding pairwise overlaps and generating a graph of all the overlaps, where each node is a read. However, just the comparison step is of complexity $O(n^2)$, and traversing the graph in such a way as to visit every node is \textit{NP}-complete. In 2001, Pevzner, et. al.\cite{5} proposed an alternative method to assembly using de Bruijn graphs. The advantage of this method is that it completely skips the overlap part, and instead breaks each read into $k$-mers, where $k$ is the read length. Instead of using reads as nodes, the method assigns them to edges, at which point the graph must be traversed such that each edge is visited once (an Eulerian path). This is solvable in linear time.

Most assemblers nowadays are not equipped to handle metagenomic assembly, however. Because closely related bacteria inevitably have large regions of homology, it is non-trivial to decide which fragments on either side of a homologous region belong to each other. Numerous methods have suggested methods to assemble bacterial genomes\cite{6,7,8}, but perhaps the most powerful such method
is Namiki and Hachiya’s MetaVelvet\textsuperscript{[9]}, which uses a modified de Bruijn graph assembly method\textsuperscript{[10]}, incorporating generation of k-mers, building a de Bruijn graph, separating chimeric paths into separate genomes based on read coverage and paired-end data support, and finally assembling scaffolds using heuristics Pebble and Rock Band\textsuperscript{[11]}. Specifically, the program finds so-called chimeric paths – ones that belong (presumably) to more than one closely related bacteria. They can be distinguished by two distinct incoming nodes, or sequences, followed by a shared homologous node, followed by two distinct outgoing nodes. Although coverage data and paired-end data are both used to infer the identity of each incoming and outgoing node, one piece of information that is not used is the similarity of the two disjoint subgraphs generated after splitting. Although these methods cope very well with assembly of scaffolds, they lack a method to find synteny of scaffolds.

To this end, I propose an extension to the method outlined in MetaVelvet called Metagenomic Assembly using Coverage and Evolutionary distance (MACE). This method uses not only difference in coverage, but also the pairwise distances between fragments to not only resolve ambiguous decisions as to the paths taken by the assembler, but also to find the synteny of contigs, defined as the colocalization of fragments within a species of bacteria. More specifically, if evolutionary distance is assumed to be constant between corresponding fragments between any two species (which should hold under certain circumstances), the distribution of distances can be used to infer the identity of each fragment.

Methods

Simulation
We start with a given number of randomly generated “parent genomes”, which serve as a basis from which to generate closely related individuals. These parent genomes contain a random number (simulated with a Poisson process) of repetitive regions substantially longer than the length of a read, and are randomly interspersed throughout the genome at random locations. These repetitive regions also have repeating elements of random length between 3 and 5. Each parent serves as a seed to create a cluster of related individuals.

For each cluster, if the number of individuals is less than the desired number, a random individual is chosen. Two uniformly chosen mutation rates for the children, \( r_1 \) and \( r_2 \), are chosen between 0.03 and 0.07. The chosen parent genome is traversed, and at each base, the probability of retaining the parent’s original base is \( 1 - r_k \), excluding repetitive and conserved regions. In both repetitive and conserved regions, no mutations are allowed. Conserved regions are randomly chosen intervals with length greater than the read length. After the two child genomes are created, the parent genome is removed. This process is repeated until we have the desired number of individuals in each cluster of genomes. Lastly, single-end reads length 50 are chosen at random positions throughout the genome with uniform likelihood. The number of reads chosen from a given individual is proportional to both its size and its abundance relative to all other genomes.

Graph Construction and Traversal
In order for the de Bruijn graph assembly method to work, we require perfect coverage. However, as the number of reads starting at each position is Poisson distributed with the parameter \( \lambda \) equal to the average coverage, there will inevitably be positions with no reads generated. Therefore, we employ a method called read splitting, in which we take a read of length 50, and generate 50 – \( k + 1 \) shorter reads from it, where \( k \) is the length of the read. For example, for a read of length 50 and \( k = 48 \), we divide the read into three reads: one with bases 1-48, the next with bases
2-49, and the last with bases 3-50. With this method, we can virtually guarantee perfect coverage, albeit at the loss of some information, i.e., the link between the first bases and the last bases.

We read through every short read in our input data and splitting it into shorter \( k \)-mers of length \( k \). The graph is implemented by taking every \( k \)-mer as an edge connecting the prefix (the first \( k-1 \) characters) to the suffix (the second \( k-1 \) characters). Because the graph is directional, we can implement it simply as a dictionary, where each prefix refers to a list of its suffixes. Because we have repeats which cause ambiguities in assembly, they are removed by testing whether the node is completely composed of \( l \)-mers, where \( l \) is between length 2 and 5.

To traverse the graph, we keep track of the number of incoming edges and outgoing edges for each node. We first search for a head node, defined as a node with no incoming edges and one or more outgoing edges. We follow any path from the head node, deleting all edges traversed and keeping track of each node visited, until we reach a node that does not have exactly one incoming and one outgoing edge. Whenever we reach such an ambiguous node, the path is broken and the resulting fragment sequence is generated.

**Figure 1A**
An example of a constructed de Bruijn graph. Each node represents a \((k-1)\)-mer, and each edge is a read of length \( k \). The unambiguous nodes are labeled blue, the ambiguous nodes are labeled red.

**Figure 1B**
The deconstructed graph from 1A. Each assembled fragments contains all of its inner nodes and every end point, which is a head node, tail node, or ambiguous node.

Finally, before we can start connecting these fragments, we must calculate each fragment's coverage. We do this by aligning each read from our original input data. First, each assembled fragment is assigned a unique integer. Then, using a sliding window of width $k$, we build a lookup table where the key is the first $s$ base of the window (in this implementation, $s = 20$), and the lookup value is the integer of the fragment and the position of the seed of length $s$. Afterward, each read from our input is fragmented into smaller reads of length $k$ to guarantee each assembled fragment can be covered, and aligned to all the fragments using the first $s$ bases as the seed, allowing no mismatches. Coverage of fragment $x$ is calculated as:

$$cvg(x) = \frac{r_k}{len(x)(m-k+1)} ,$$

where $r_k$ is the number of read fragments of length $k$ that align to the segment and $m$ is the length of the original read. We divide by the factor $(m-k+1)$ because this is the number of smaller reads each read is fragmented into, though for purposes of constructing paths and joining scaffolds, this term can be omitted.

Splitting Chimeric Regions
Once the set of unambiguous fragments and their coverage has been generated, we find overlaps of $k-1$ letters between each pair of fragments and generate a graph, where each fragment acts as a node, and each directed edge is an overlap of the last $k-1$ letters of the first fragment and the first $k-1$ letters of the second fragment (the overlap-layout-consensus method). Because we have cut the search space from a few hundred million reads down to a few thousand fragments, we are not worried about the time complexity of this step. Chimeric fragments are defined as fragments that have at least one end with two or more connections (Figures 2A-2D). Chimeric nodes can be head nodes, tail nodes, or internal nodes.

![Figure 2](image)

**Figure 2**
The red nodes are chimeric nodes, each of which is composed of one or more paths. In 2A, the chimeric node will most likely be decomposed into two paths, as it is most likely a head node. In 2B, the chimeric node most likely only takes one of the two possible outgoing paths, as the other node connected by an outgoing path can be accounted for by the homologous segment below the chimeric segment. In 2C, the chimeric node is expected to be the intersection of two subgraphs, and the precise path must be found. Lastly, in 2D, any number of the outgoing and incoming edges may be used, so we enumerate all possibilities and calculate the most likely path.

Our goal is to decompose each chimeric node into a collection of solitary nodes, which will constitute our collection of contigs. We begin by traversing through our graph in any order until we reach a
chimeric node. In the case where the incoming side has none or one connecting edges and the outgoing side has two or more, there are several possibilities. First, it is possible only one of the outgoing paths is the correct one (Figure 2A, 2B). If this is the case, we expect the coverage of exactly one of the nodes connected to outgoing edges to be equal to the coverage of the chimeric node. In the second possibility, the chimeric node may be composed of two paths which split at the end of the chimeric node. In this scenario, we expect the sum of the coverages of the nodes connected to outgoing edges to be approximately equal to the coverage of the chimeric node. Lastly, if there are three or more outgoing edges connected, there may be up to that many paths passing through the chimeric region. To figure out which is the correct scenario and path(s), we take the power set of all nodes connected on the outgoing end and compare the sum coverage of each set within the power set to the coverage of the chimeric node. We calculate the error as:

\[
error(i) = \frac{\left|cvg(\alpha) - \sum_{j=1}^{n}(cvg(\beta_j))\right|}{(cvg(\alpha) + \sum_{j=1}^{n}(cvg(\beta_j)))/2}
\]

where \(i\) is the set of nodes passing through the chimeric region \(\alpha\), and \(\beta_j\) is the outgoing fragment with index \(j\). We then take the set \(i^*\) with lowest error, and if it is less than our threshold of 0.15, and assemble the path. For each path predicted to pass through the chimeric node, we add a new fragment to our collection with the sequence of the chimeric region and the outgoing node connected through the path. We estimate the coverage of each new node as the coverage of the original outgoing node. Lastly, the original chimeric region and the used nodes connected by outgoing edges are all removed from our set of fragments. The same process, but reversed, is used for resolving scenarios in which there are none or one outgoing edges and multiple incoming edges.

The more complicated case involves two or more incoming outgoing edges, where the number of incoming edges and outgoing edges is equal (Figure 2C). In other words, the node is balanced. If we label the incoming nodes as \(\alpha_n, \alpha_{n-1}, ..., \alpha_1\) and the outgoing nodes as \(\beta_m, \beta_{m-1}, ..., \beta_1\) the coverage and sequence of each becomes important in deciding which incoming edge belongs to which outgoing edge. There are \(n!\) ways of matching each \(\alpha_i\) to a \(\beta_j\), so we find all \(n!\) permutations of \((\beta_m, \beta_{m-1}, ..., \beta_1)\) and label the \(i^{th}\) permutation as \((\beta_{i,m}, \beta_{i,m-1}, ..., \beta_{i,1})\). For each permutation \(i\), the total error \(e(i)\) is calculated as:

\[
error(i) = p \sum_{j=1}^{n} \frac{|cvg(\alpha_j) - cvg(\beta_{j,i})|}{(cvg(\alpha_j) + cvg(\beta_{j,i}))} + (1 - p) \sum_{k=1}^{n} \sum_{l=k+1}^{n} \left[ d(\alpha_k, \alpha_l) - d(\beta_{k,i}, \beta_{l,i}) \right].
\]

Here, \(p\) is a parameter that controls the amount of weight given to the coverage difference versus the amount of weight given to the differences in distance. The distance \(d(\alpha, \beta)\) is defined as the Levenshtein distance between \(\alpha\) and \(\beta\), but modified as to only calculate it for the first \(m = \min(\text{len}(\alpha), \text{len}(\beta))\) bases, and aligned either from the left if an outgoing edge or on the right if an incoming edge. It is then divided by the maximum distance of two strings so as to obtain a distance in \([0,1]\), where 0 is the distance of two identical strings and 1 is totally dissimilar strings. To decrease the memory footprint from \(O(\text{len}(\alpha) \times \text{len}(\beta))\) to \(O(\min(\text{len}(\alpha), \text{len}(\beta)))\), we can employ the Wagner-Fisher algorithm\(^{12}\). To further simplify the time of calculation, an arbitrary maximum distance of \(m/4\) is imposed for the Ukkonen algorithm\(^{13}\), where the constant of \(\epsilon\) can be any constant desired. As the expected edit distance between two unrelated string remains an unsolved problem, but estimated to be just over half of the length of two equal length strings, a reasonable maximum for the constant is 2. Once the error for each permutation is calculated, the lowest error
is taken as the optimal permutation, and the corresponding segments $\alpha$ and $\beta$ are connected through the chimeric region. The process of generating new fragments is identical to the case of resolving decisions on one side, except that fragments on both sides are taken to constitute the new fragments, and the new coverage is calculated as the average of the coverages of connected nodes.

The last and most complicated case is where the number of multiple incoming and multiple outgoing edges is unequal. In this scenario, we hypothesize that it is due to another consecutive region of homology on the side with less connections (Figure 2D). As such, if we temporarily collapse nodes on the side with more connections in the correct combination, we will be able to reduce the problem to the case where the node is balanced. We take every possible clustering of the $n$ distinct edges into $r$ indistinct bins, where each bin has at least one member. We can enumerate the possibilities using a recursive combinatorial method, and the total number of clusterings is the Stirling number of the second kind, denoted $\{n\}^{[14]}$. With each clustering, we then permute the clusters and calculate the error as described above. Again, we form new fragments, but include a fragment for each path within collapsed edges as well.

We impose an additional restriction in resolving ambiguities: any case with minimum error of over 1 (generally obtained with multiple incoming and outgoing edges) is discarded.

Joining Scaffolds
The last step in my method is to cluster assembled contigs with each other, also known as synteny, with the assumption that these fragments come from the same organism. A similar method to the one described above is used. First, the distance between every pair of fragments is calculated. Because of the reasons outlined above, most pairs of fragments will have the maximum distance of 1. As the edit distance is very sensitive even to much-diverged sequences (sequences with up to 50% mutation between them are detectable), we can expect to discover homologous regions robustly under the assumption that there are not many transposed elements in the same order. Any fragments with a distance of less than 1 between each other are clustered together, and form $g$ disjoint clusters. All clusters are then compared to each other pairwise. Using the same notation as above, if we have one cluster with fragments $\alpha_1, \alpha_2, ..., \alpha_n$ and another cluster with fragments $\beta_1, \beta_2, ..., \beta_n$, we permute the second cluster $n!$ ways. For each permutation, a score is calculated as the sum of the differences in coverage between each pair and the sum of all pairwise distances. If the minimum of the scores below a certain threshold, the permutation is considered to be the “true” synteny of the contigs. The threshold can either be chosen by the user or assumed to be similar to the minimum scores in the chimeric region separation phase, as those have a much higher likelihood of being true positives.

Results
To test the method, I simulated clusters of genomes with the following parameters: average genome size, the number of separate trees or families of organisms, the number of children or members in each family, and lastly, the number of sequencing reads. Throughout the whole experiment, for simplicity and sake of testing the efficacy of the method, I used only single-end reads of length 50 base pairs with no errors. We would expect results to be favorable for paired-end reads and for longer reads, but worse with errors. However, because of the scope of the project, errors were not accounted for in the method.

I compared the method with looking at pairwise distance versus without across the parameter space. Without looking at evolutionary distance, the method is essentially the same as that
described in MetaVelvet\cite{9}. Because we use distance to both separate chimeric fragments into their components and to compare contigs to infer synteny, we expect the results using MACE to give us more confidence in both, resulting in both longer fragments with which to assemble contigs, and better bottom-up synteny prediction. For performance metrics, I used time of assembly (which we expect to be longer for MACE due to calculation of evolutionary distance, though it was optimized in several ways), the total coverage of the target genomes, the estimated abundance of the genome, and the LD50 of the genome. I also looked at the number of clusters of fragments predicted to belong to the same organism that actually belonged to the target genome. With perfect synteny prediction, this number would be 1, as all the assembled contigs that belong to the target genome would be binned together by MACE. Without any prediction, this number can be as high as the total number of contigs in the target genome. Lastly, the LD50 is defined as the length of the shortest contig such that the sum of its length and all longer contigs constitute at least 50% of the target genome. If 50% coverage of the target genome is not attained, LD50 is defined to be 0.

For the baseline parameters, I used simulated genomes of approximately 100,000 base pairs, each with approximately 20 repetitive regions of length 500 base pairs, three families of bacteria, three children within each family, and 100 million reads. From that baseline, I simulated varying the following parameters: approximate genome size, the number of reads generated, the number of trees of families, the number of members in each family, and the relative abundances of the members of the families. With these varying parameters, I compared the method with and without finding pairwise evolutionary distance, and measured accuracy of results and time of calculation. All calculations were performed on a Unix based machine with a 2.6 GHz processor and 16 GB of RAM and flash storage.

As expected, contig construction takes longer taking under account both distance and coverage as opposed to only coverage. Synteny cannot be predicted based on only coverage, as any contigs with equal coverage would not be separable between organisms. Surprisingly, calculating distance for
means of separating chimeric nodes does not take as long as expected, amount to only 33 minutes of
addition computing time. The additional calculation do, however, slightly increase accuracy:

The most marked difference arises in the comparison of the total number of fragments per genome.
This is defined as the average number of “organisms” which make up the true target genome. In the
method that uses distance as a predictor, the average number of fragments per genome drops from
27 to 2.3. This shows us that under the assumptions given, we can reliably predict which fragments
belong to the same genome. Additionally, for a given genome, most contigs are clustered together, at
86%, while the remaining 14% of contigs are all in individual clusters. Upon further inspection,
these ended up being contigs that had either mutated much more than expected or were constructed
so as to align with the centers of the homologous contigs. This exposes a flaw in the implementation
of the evolutionary distance calculation algorithm – it assumes that the two fragments are aligned
at either end, and not in the middle. To this end, we can modify the algorithm to allow for local
alignments, but at the cost of longer computing time.

There were many additional parameters modified, but due to space constraint, only relevant ones
are presented. The average size of the genomes and the number of genomes in each family made the
greatest difference in accuracy:
As to the number of genomes per family, it made the largest difference in accuracy. This is as predicted, as the greater the number of genomes, the more likely it is that multiple genomes will be both similar in sequence and have similar coverage. In these scenarios, the additional calculation of pairwise evolutionary distance makes the greatest difference:
Discussion

There were several assumptions made in this study that may or may not be realistic, depending on the circumstances. First is the assumption that the genomes are closely enough related that they retain a significant amount of shared sequence. Even more so is the assumption that all parts of the genome mutate at approximately the same rate. While this may be true in a large enough scale, this definitely does not hold when transpositions, insertions, deletions, and inversions are considered. However, this can theoretically be accounted for, mostly with modification of the distance function. An example of this is the introduction of a penalty term at the beginning of an insertion or deletion, but then eliminating the penalty term with every consecutive base inserted or deleted. Another example is to do only a local alignment, but this requires a much larger memory space. This is particularly useful for inversion regions and for transposons, which are quite common in bacteria\cite{15}.

The assumption of uniform mutation is also generally not true, but there are several ways to account for this. Firstly, it is not necessary to have uniform mutation to find correlated regions, as edit distance is very sensitive to even small similarities. Secondly, there are multiple ways to find evolutionary distance in non-uniformly mutating regions. For example, if we know the open reading frame (either from homology to already-sequenced bacteria or prediction via e.g., GLIMMER\cite{16}), we can use third position degeneracy of codons to obtain a reasonable estimate for evolutionary distance, under the hypothesis that there is no selective pressure on degenerate positions. MACE can be modified relatively easily to account for such situations.

A disadvantage of this method is the computation time of all the distances. Using the standard Wagner-Fisher algorithm for evolutionary distance\cite{12}, both time of computation and memory space necessary are $O(n^2)$, and there are $O(n^2)$ total sequence comparisons. However, with a few adjustments and assumptions, we can decrease both. Firstly, if we do not need to trace back our alignment path, we do not need to store the entire $n^2$ matrix. We only need to store the previous array and the current one. This reduces memory space to $O(n)$. Secondly, if we assume some maximum distance, only values within that distance need to be calculated in the matrix. If we assume the maximum distance is small, it can greatly improve our calculation time, realistically up to an order of magnitude. Lastly, if we know the sequences should be aligned on one side, if after a certain number of bases (e.g., 500) the distance is above a certain number (e.g., 200), we can terminate the process and return the maximum distance. The more we know \textit{a priori} about the sequences, the more we can optimize our computation time and memory.
The results suggest that in the simplified cases where the abundances of the strains within a family are different, inclusion of a distance metric does not give us significantly better results, and only increases the computing time. However, the results also suggest that in the case where we have many similar abundances amongst similar genomes, evolutionary distance acts as a sort of “tie-breaker”. As far as clustering contigs into predicted genomes, doing so by coverage alone is very risky, as any strains of similar coverage will be clustered together. Considering how much coverage of individual fragments can vary within a single organism, it is difficult to ascertain how well such a method would perform.

Because we have the additional information of genetic sequence from related strains, we can infer which contigs belong to the same genome. Although it was not implemented here, it is also possible to look not only for clusters of sequences with similar distance and coverage distributions, but also from subclusters. For example, if we have a cluster of similar sequences $A_s$, $A_s$ and $A_s$, and another cluster with sequences $B_s$ and $B_s$ if we find that the distance between $B_s$ and $B_s$ and their relative coverages are similar to that of $A_s$ and $A_s$, we can infer synteny between $B_s$ and $B_s$ and between $B_s$ and $A_s$. Additionally, we have the added result that there may or may not be a region missing in $B$ which correlates with $A_s$, or that the region in $B$ has multiple copies in the organism, leading to an increased predicted coverage.

References: