Genome assembly

(for more info see: http://www.cbcb.umd.edu/research/assembly_primer.shtml)

Introduction

Sequencing technologies can only "read" short fragments from a genome. Reconstructing the entire sequence of the genome, thus, requires that these fragments be joined together in a jigsaw-puzzle-like process. Note that, in order for the reconstruction to even be possible, the individual sequences must be sampled from random locations in the genome. Also, enough sequences must be sampled to ensure that the individual sequences overlap, i.e. enough information is available to decide which sequences should be joined together. The process through which the sequences are generated is called "shotgun sequencing", and involves the random shearing (through a physical process) into small fragments of a collection of copies of the genome of interest.

Is assembly possible: Lander Waterman statistics

Note that it is not even clear that the assembly of a genome from small pieces should even be possible. Given that the process through which the sequences are generated is random, it is possible that certain parts of the genome will remain uncovered unless an impractical amount of sequences are generated.

To assess the theoretical feasibility of the assembly of shotgun sequencing data, Eric Lander and Mike Waterman developed a statistical analysis based on Poisson statistics. Briefly, if some events occur uniformly at random (e.g. the start of a sequencing fragment along a genome can be assumed to be chosen uniformly at random), the number of events occurring within a given time interval is represented by a Poisson distribution.

Given an average "arrival rate" λ (# of events occurring within a given interval of time), the probability that exactly n events occur within the same interval is expressed by the formula:

$$f(n,\lambda)=\frac{\lambda^n e^{-\lambda}}{n!}$$

In the context of sequencing we are interested in finding intervals that contain no events (n=0) - these would represent gaps in the coverage of the genome by sequences.

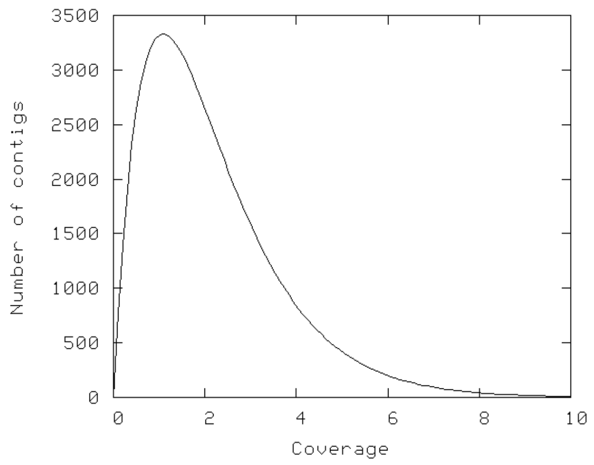
The Lander-Waterman statistics estimate the number of gaps in coverage (conversely the number of contiguous DNA segments) that can be expected given the following set of parameters:

- G - genome length
- n - number of sequences
- L - length of sequences
- c = nL/G - depth of coverage (number of times genome over-sampled by the set of

sequences)

- t - the amount by which two sequences need to overlap in order to computationally detect this overlap
- $\sigma = (L-t)/L$

Among other numbers, the L-W statistics provide estimates for the expected number of contigs: $n e^{-c\sigma}$



Drawing 1: Lander-Waterman: expected # of contigs given coverage

As can be seen from the figure above, the expected number of contigs rapidly decreases once coverage exceeds about 8-10-fold, i.e. after over-sampling a genome by about 10 times, the assembly should be theoretically possible.

Shortest common superstring and greedy algorithm

A simple formulation of the assembly problem as an optimization problem phrases the problem as the Shortest Common Superstring Problem: find the shortest string that contains all the sequences as substrings. In other words, find the most parsimonious "explanation" for the set of sequences. A fair amount of work went into this problem in the 80s-90s - the problem was shown to be NP-hard, and a series of approximation algorithms were developed that approach an approximation factor of 2 (the reconstructed string is at most twice as long as the optimal).

A simple greedy algorithm can be proven to yield a 4-approximation, however it is conjectured that this algorithm is actually 2-optimal, given that no example has yet been found where the greedy assembly algorithm has generated a worse approximation.

The greedy assembly algorithm proceeds as follows:

1. compare all sequences in a pairwise fashion to identify sequences that overlap each other.
2. pick the sequences that overlap each other the best and merge them

3. repeat step 2 until no more sequences can be merged, or the remaining overlaps conflict with existing contigs.

While this algorithm is only an approximation, it has been extremely successful in practice - most early genome assemblers (phrap, TIGR Assembler, CAP) relied on this simple greedy heuristic, and were successful in reconstructing a range of genomes.

Graph formulations for assembly

The parsimony definition of assembly implicit in the SCS problem can be easily seen not to be relevant in a biological setting, primarily due to the presence of repeated DNA sequences (repeats) within the genomes of most organisms. These redundant sequences would be collapsed into a single "unit" by any algorithm that attempts to solve the SCS problem.

Instead other optimization criteria have been proposed that attempt to capture the biological nature of the problem. Myers proposed, for example, that we should phrase the assembly problem as the task of reconstructing a layout of the sequences that is consistent (in terms of Kolmogorov statistics) with the characteristics of the random process that generated the sequences. Unfortunately this formulation is hard to translate into a practical algorithm, though it is important to keep in mind especially in the context of validation.

Instead, most modern assemblers formulate the assembly as a graph traversal problem.

Overlap-layout-consensus/string graph

A first formulation creates a graph that represents each sequence as a separate node, and creates an edge between any two nodes whose corresponding sequences overlap. In this formulation, we want to find a traversal of the graph that uses all the sequences, each exactly once (you cannot use a same sequence in multiple places in the genome), i.e. we are looking for a Hamiltonian path - a well known NP-hard problem.

Recently, Myers has shown that through a few simplifications, including the removal of transitive edges, the problem can be rephrased as a Chinese Postman problem (find the shortest tour that traverses each edge in a graph at least once) - a problem that can be solved in polynomial time. The transformed graph is termed the *assembly string graph,* and the paper describing this concept is listed on the course website.

DeBruijn graph

An alternative formulation for the assembly problem arose form early explorations of a sequencing technology called "sequencing by hybridization". In this approach, one could find out if a given k-mer occurred in the genome being sequenced, i.e. the assembly problem could be phrased as:

Given the collection of **all** k-mers (strings of length k), reconstruct the genome sequence.

In some sense, this problem is equivalent to a shotgun process that generates reads of length exactly k, and which guarantees that exactly one read starts at each position in the genome (perfect coverage).

Pavel Pevzner formulated this problem in the context of deBruijn graphs: a deBruijn graph of order k is a graph that contains all strings of length k-1 from a given alphabet as nodes, and contains an edge between two nodes if the corresponding strings overlap by exactly k-2 letters. In other words, each k-mer in the genome is represented by two nodes in the graph connected by an edge. In the context of assembly we are looking at the subgraph of the complete deBruijn graph that contains just the k-mers present in the genome (as inferred from the set of reads).

As we are trying to find an assembly that contains all the k-mers of the genome, we are looking for a path through the graph that visits every edge at least once (edges visited multiple times are repeats), i.e. a Chinese Postman path. Note that if we can infer from experimental data (e.g. from the number of sequences containing a particular k-mer) the number of times we need to visit each k-mer, the problem becomes NP-hard - we are looking for a Chinese Postman path of a given length.

Complexity results

The formulation of assembly as a variant of the Chinese Postman problem implies that the assembly problem can be solved in polynomial time. This is only partly true - a solution to the assembly problem can be found in polynomial time, however there are an exponential number of possible solutions, i.e. the problem is underconstrained. Adding additional constraints, such as adding information about the multiplicity of repeats, generally results in an NP-hard variant of the problem.

Just how many possible Eulerian tours are in a graph (assuming an Eulerian graph, e.g. as generated from the Chinese Postman path by duplicating certain edges)? The answer can be fairly easily computed for directed graphs (though it's intractable in undirected graphs). The reasoning is based on a simple algorithm for finding an Eulerian tour:

1. Pick an arbitrary node in the graph n
2. Construct a directed spanning tree having n as a root, such that all the edges are pointing upwards (the tree is a collection of directed paths connecting the leaves with the root)
3. Start traversing the graph in a greedy fashion, starting from n - after entering a node, leave the node using an edge that does not belong to the spanning tree, if possible. In other words, whenever a node is encountered on the traversal of the graph, the unique spanning tree edge leaving this node (linking the node to its parent) is traversed last.

This algorithm completes only after finding an Eulerian tour - the requirement that the tree edge is the last visited ensures that there is always an "out", thus the algorithm cannot get stuck before visiting all the edges in the graph.

Thus, the number of possible tours can be computed as a function of two terms: (i) the number of spanning trees in the graph (see notes from Sam on how to compute this number); and (ii) the different possible traversals of the graph once the spanning tree is fixed. The latter is simply the product of all different permutations of the edges exiting each node in the graph (minus the spanning tree edge), i.e.:

$$N_{Euleriantours} = N_{spanning\ trees} \prod_{v \in V(G)} (outdeg(v) - 1)!$$

Theory vs. practice

As shown above, the assembly problem can be shown to be NP-hard under a number of formulations. However, these theoretical results contradict the empirical observation that even simple heuristic algorithms can generate correct assemblies in most situations. I.e. the instances encountered in practice are not necessarily complex.

Niranjan Nagarajan and myself tried to more precisely determine the boundaries of this disagreement between theory and practice, specifically we parametrized the assembly complexity as a function of the relationship between read length (L), overlap length (o), and the length of the repeats (R) found in the genome. Below is a quick summary of the results. For full details see the paper (linked from the syllabus site).

Case 1: R < o - repeats do not confuse the assembly, any algorithm will find the unique correct solution.

Case 2: R > 2L-o - the assembly problem is equivalent to the Chinese Postman problem - a solution (not necessarily the correct one) can be found in polynomial time. Also, in this case a solution to the Shortest Common Superstring problem is not a solution to the assembly problem.

Case 3: o <= R <= 2L-o - the solution to the assembly problem is the same as the solution to the SCS problem. Assembly is NP-hard.

Note, that as a corollary, if sufficient coverage is generated to ensure that o = L-1, the third case disappears, i.e. the assembly is either trivial, or underconstrained, but not NP-hard.