Genomics and Bioinformatics: Week 2

24 September 2013

1 Overlap graphs

The data available is a set of N sequence reads $R_1, \ldots, R_N$ each of length L. A read R_j overlaps a read R_k (in this order) if the last ℓ bases of R_j match the first ℓ bases of R_k, for example ($L = 14$, $\ell = 10$):

$$R_j = \text{ACGT}\underline{\textbf{GTCCGATTGG}}\ , \qquad R_k = \underline{\textbf{GTCCGATTGG}}\text{TGTA}\ .$$

The **overlap graph** is constructed as follows: its vertices are the reads $R_1, \ldots, R_N$ and it contains a directed edge from the vertex R_j to the vertex R_k if R_j overlaps R_k:

$$R_j \xrightarrow{\text{GTCCGATTGG}} R_k\ .$$

Any path in the resulting graph not visiting the same vertex twice generates a sequence contig. We are naturally interested in finding the longest such path. We can then remove the visited vertices and restart on the remaining graph.

2 Hamiltonian path problem

If we assume that all reads are error-free and that there is no ambiguity, then we can formulate a general problem as follows:

Definition 1. *A **Hamiltonian path** in a graph is a path visiting every vertex once and only once.*

This is a well-known problem in computer science that is unfortunately known to be "untractable" meaning that finding an algorithmic solution that scales well with the number N of vertices is impossible.

We will therefore change the formulation of the problem to make it simpler to solve.

3 Eulerian path problem

Let us construct the *dual graph* to the overlap graph: this corresponds to replacing edges by vertices, as follows: make one vertex for each ℓ-mer present in the read set (each subsequence of length ℓ). Then make an edge from V_j to V_k if there is a read in the set starting with V_j and ending with V_k:

$$R_j = \text{ACGTGTCCGATTGG}\ , \qquad R_k = \text{GTCCGATTGGTGTA}\ .$$
$$\text{ACGTGTCCGA} \xrightarrow{R_j} \text{GTCCGATTGG} \xrightarrow{R_k} \text{GATTGGTGTA}\ .$$

We keep a table of which reads are associated with every edge. Then a contig corresponds to a path in the graph not visiting the same edge twice:

Definition 2. *An **Eulerian path** in a graph is a path visiting every edge once and only once. If the path closes on itself it is an **Eulerian cycle**.*

Theorem 1. *There exists an **Eulerian cycle** in a graph if and only if every vertex in the graph is **balanced**, namely the number of incoming edges is equal to the number of outgoing edges:*

$$\forall\ k\ :\quad \text{indegree}(V_k) = \text{outdegree}(V_k).$$

This is simple to understand: suppose that one vertex is unbalanced (*e.g.* $\text{indegree}(V_k) < \text{outdegree}(V_k)$), then at one point all incoming edges will have been visited by the path, but not all outgoing edges. The path will never come back to this vertex anymore (because it cannot use the same incoming edge twice), so these unvisited outgoing edges will never be included in the path which will therefore not be *Eulerian*.

Conversely suppose that the graph is balanced. Start from any vertex and follow edges randomly until you come back to the same vertex. If some vertex along that path still has unvisited edges, then you start a new path from that vertex until it closes on itself. These two cycles can be merged into one by switching the order in which the edges are visited so as to follow the second cycle before continuing along the first. Iterate this until there is no unvisited edge left.

There will be an **Eulerian path** in a graph if and only if you can add an edge connecting the endpoint of the path to its start and the resulting graph is balanced.

In practical situations, the graph built on real reads will not be balanced and there will be many ambiguities in overlaps due to sequencing errors. The motivates the following extension of the problem.

4 de Bruijn graphs

We describe here the algorithm suggested by Pevzner et al. and implemented in the software *EULER*.

First we make a list of all ℓ-mers present in the set of reads, and then construct the graph as above: for each ℓ-mer in the list we draw an edge from the node V_j which represents its $\ell - 1$ first nucleotides to V_k representing its $\ell - 1$ last nucleotides.

$$\begin{aligned} & \text{ACGTGTCCGA} \xrightarrow{R_j} \text{CGTGTCCGAT} \xrightarrow{R_j} \text{GTGTCCGATT} \xrightarrow{R_j} \\ \xrightarrow{R_j} \ & \text{TGTCCGATTG} \xrightarrow{R_j} \text{GTCCGATTGG} \xrightarrow{R_k} \text{TCCGATTGGT} \xrightarrow{R_k} \\ \xrightarrow{R_k} \ & \text{CCGATTGGTG} \xrightarrow{R_k} \text{CGATTGGTGT} \xrightarrow{R_k} \text{GATTGGTGTA} \ . \end{aligned}$$

Remark that there can be be several parallel edges connecting the same two vertices, and that we also must include the reverse complements of each read, *e.g.*:

$$\text{CCAATCGGAC} \xrightarrow{\overline{R_j}} \text{CAATCGGACA} \xrightarrow{\overline{R_j}} \text{AATCGGACAC} \xrightarrow{\overline{R_j}} \dots$$

We then correct for sequencing errors by introducing up to Δ mutations in a read whenever these mutations have the effect of reducing the total length of the ℓ-mers list. For example, adding the read $R_i = \text{TTACGTCTCCGATT}$ to the graph would create 5 new nodes, without connection with another read. However, allowing $\Delta = 1$ mutation we can reduce this to only 2 new nodes:

$$\begin{aligned} & R_i \ = \ \text{TTACGT}C\text{TCCGATT} \Rightarrow \text{TTACGT}G\text{TCCGATT} \ , \\ & \text{TTACGTGTCC} \xrightarrow{R_i} \text{TACGTGTCCG} \xrightarrow{R_i} \text{ACGTGTCCGA} \xrightarrow{R_i,R_j} \\ \xrightarrow{R_i,R_j} \ & \text{CGTGTCCGAT} \xrightarrow{R_i,R_j} \text{GTGTCCGATT} \xrightarrow{R_j} \text{TGTCCGATTG} \xrightarrow{R_j} \dots \end{aligned}$$

In general, erroneous reads create either "bulges" or "dead ends" in the graph, which can be recursively removed by this process.

We finally enumerate all paths in this graph that never visit the same edge twice (with multiplicity). By mapping back the full (uncorrected) reads on these contigs, we can compute the consensus sequence and the possible polymorphisms.

5 Using paired-end reads

When both ends of a DNA fragment are sequenced and provided as a linked pair, this is called *paired-end reads*. Knowing the (approximate) length of the fragment, hence the probable distance along the genome separating the two paired reads, we can use this information to constrain the assembly process. Then if several paths in the graph connect two reads from the same pair, the one with length closest to the expected fragment size will be kept and the other paths excluded. This greatly reduces ambiguities in the graph and therefore increases contig lengths.

Reference

1. Pevzner, P. A., Tang, H., & Waterman, M. S., *An Eulerian path approach to DNA fragment assembly*, PNAS **98**, 9748–9753 (2001).