10 Sequence assembly

This chapter is based on the following papers, which are recommended reading:


Despite a lot of work done in the area, sequence assembly continues to be a challenging practical problem.

**Assembly problem:**
**Input:** Set of short DNA reads from a genome
**Goal:** Determine complete genome sequence

Usually, reads are assembled into a number of larger contiguous sequences, called *contigs*:

\[
\text{DNA reads} \quad \text{assembly} \quad \text{contigs}
\]

10.1 Overlap-Layout-Consensus

There are two main approaches to sequence assembly, namely

- the *Overlap Layout Consensus* approach (OLC), and

- the de Bruijn graph approach.

The OLC approach was pioneered by the “Celera assembler”, which was originally developed at Celera Genomics Corp. to assemble

- the *Drosophila melanogaster* genome (2000), and

- the Human genome (2001).

Here is a sketch of the steps involved:

\[
\text{DNA reads} \quad \text{Overlap computation} \quad \text{layout} \quad \text{consensus}
\]

- pairwise alignments

- spans tree in overlap graph

- multiple alignment and consensus
In this approach:

- Compute pairwise alignments of all pairs of reads.
- Represent each read by an edge (or, more compactly, by a node) in graph.
- Represent each overlap by an edge.

As the size of datasets increase, there are two main challenges:

- How to determine all pairwise overlaps of reads?
- How to limit the size of the graph used to represent the data.

(We discussed this approach in detail in *Grundlagen der Bioinformatik*.)

### 10.2 de Bruijn Graph

The de Bruijn graph is a datastructure that is used to represent sequencing data and is used in sequence assembly.

It is defined as follows.

**Definition 10.2.1 (de Bruijn Graph)** Let $R = \{f_1, \ldots, f_t\}$ be a set of sequencing reads. Choose a fixed odd number $k$. The de Bruijn graph $G = (V, E)$ has node set $V$ that consists of all different $k$-mers that (words of length $k$) appear in $R$.

Two nodes $v$ and $w$ are connected by a directed edge $(v, w) \in E$ if the last $k - 1$ letters of $v$ match the first $k - 1$ letters of $w$.

**Example:**

Assume we are given $R = \{f_1, \ldots, f_4\}$, with

- $f_1 = \text{TGAC}$
- $f_2 = \text{ACTG}$
- $f_3 = \text{TTGAC}$
- $f_4 = \text{GATTGCC}$

Then the de Bruijn graph for $k = 5$ looks like this:

![De Bruijn Graph Diagram](image)

**10.2.1 Simplifying the de Bruijn graph**

We now introduce two simplifications to the de Bruijn graph.
(1) We introduce block nodes that represent a chain of \( k \)-mers that all come from the same read, keeping the last letter of each \( k \)-mer:

(2) We simultaneously maintain the reverse complement of all \( k \)-mers:

10.3 Computation the de Bruijn graph

Input: set of reads \( R \), fixed parameter \( k \).

For example, let \( k = 5 \) and \( R \) be the following reads:

\[
\begin{align*}
\text{f}_1 & = \text{TAGACTG} \\
\text{f}_2 & = \text{ACTGATTG} \\
\text{f}_3 & = \text{ATTGACCA} \\
\text{f}_4 & = \text{GATTGCC} 
\end{align*}
\]

The graph is built in a number of steps:

1. Compute a mapping of \( k \)-mers to their first occurrences
2. Rewrite reads as chains of \( k \)-mers
3. Reverse direction of references to get “road map”
4. Build graph from road map

10.3.1 Map \( k \)-mers to first occurrence

We first compute a table mapping \( k \)-mers the location of their first occurrence in the list of reads:

<table>
<thead>
<tr>
<th>( i )</th>
<th>( k )-mer</th>
<th>first occurrence (read, pos)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TAGAC</td>
<td>(1,1)</td>
</tr>
<tr>
<td>2</td>
<td>AGACT</td>
<td>(1,2)</td>
</tr>
<tr>
<td>3</td>
<td>GACTG</td>
<td>(1,3)</td>
</tr>
<tr>
<td>4</td>
<td>ACTGA</td>
<td>(2,1)</td>
</tr>
<tr>
<td>5</td>
<td>CTGAT</td>
<td>(2,2)</td>
</tr>
<tr>
<td>6</td>
<td>TGATT</td>
<td>(2,3)</td>
</tr>
<tr>
<td>7</td>
<td>GATTG</td>
<td>(2,4)</td>
</tr>
<tr>
<td>8</td>
<td>ATTGA</td>
<td>(3,1)</td>
</tr>
<tr>
<td>9</td>
<td>TTGAC</td>
<td>(3,2)</td>
</tr>
<tr>
<td>10</td>
<td>TGACC</td>
<td>(3,3)</td>
</tr>
<tr>
<td>11</td>
<td>GACCA</td>
<td>(3,4)</td>
</tr>
<tr>
<td>12</td>
<td>ATTGC</td>
<td>(4,2)</td>
</tr>
<tr>
<td>13</td>
<td>TTGCC</td>
<td>(4,3)</td>
</tr>
</tbody>
</table>
10.3.2 Rewrite as chains

We then rewrite reads as chains of $k$-mers, with reference to the first occurrence of each $k$-mer:

- $f_1 = 1, 2, 3$
- $f_2 = 4, 5, 6, 7$
- $f_3 = 8, 9, 10, 11$
- $f_4 = 7, 12, 13$
- $f_5 = 6, 7$

The result is called the road map, which contains all information necessary to construct a de Bruijn graph with block nodes.

10.3.3 Reverse references

We then reverse the direction of all references:

- $f_1 = 1, 2, 3$
- $f_2 = 4, 5, 6, 7$
- $f_3 = 8, 9, 10, 11$
- $f_4 = 7, 12, 13$
- $f_5 = 6, 7$

10.3.4 Build graph from road map

- Each maximal chain of $k$-mers that does not contain the source node of some reference edges defines a block node in the graph.
- Block nodes that are consecutive in the road map are linked by an edge in the graph. Reference edges give additional links:

  

Note that this resulting graph is not connected, because no reads spans the positions 3 and 4, or the positions 7 and 8. (This is different than the graph shown above as an example of a de Bruijn graph because here we only connect two $k$-mers if they have an overlap of length $k - 1$ and this overlap occurs in at least one read.)

10.3.5 Simplification of graph

A simple path of nodes can be merged into a single block:
10.4 Dealing with imperfect data

On real data, the algorithm as described so far will produce a huge, ugly graph. Any assembler requires additional steps to handle real data, because:

- Sequencing errors cause replacement, insertion or deletion of bases in reads.
- Biological features such as polymorphisms (e.g. diploid organisms) also cause discrepancies.

Error removal algorithms aim at resolution of discrepancies to simplify assembly. The main idea is to identify errors via topological features of graph.

10.4.1 Error removal algorithms

There are a number of algorithms that aim at simplifying the graph by removing nodes and edges that may be due to errors:

1. Tip removal
2. Bubble removal
3. Removal of erroneous edges
4. Use of mate pairs

10.4.2 Tip removal

The number of reads that span an edge in the de Bruijn graph is called the multiplicity of the edge. A tip is a chain of edges that branches off from a longer chain of edges in the graph. In this example, the edges are labeled by their multiplicity:

The tip removal heuristic is based on the following ideas:

- A tip should be short, e.g. \( \leq 2k \).
- Its starting edge must have lowest multiplicity among all edges leaving the same node.
- Repeatedly remove tips and then simplify (as described above).

In the above example, the edge leading to the node with label GTAT may be considered a tip.
10.4.3 Bubble removal

A bubble consists of two paths that start and end at the same nodes, respectively, and have similar sequence.

They are created by errors or biological variants (edges numbered here in BFS order):

\[
\begin{array}{cccccc}
A & \xrightarrow{3} & B & \xrightarrow{4} & C & \xrightarrow{8} & E \\
& & X & & & \\
& & 1 & & 1 & & \\
B' & \xrightarrow{1} & C' & \xrightarrow{2} & & & \\
\end{array}
\]

Bubbles are removed using the tour bus algorithm:

Do a breath first search from node A, using the following depth function:

\[
d(v, w) = \frac{\text{length}(w)}{\text{multiplicity}(v, w)}
\]

Assume that we reach node w from A via a path \( p_1 \) in the BFS.
Assume that we later reach w again, via some other path \( p_2 \).
If \( p_1 \) and \( p_2 \) represent similar sequences, then we merge \( p_2 \) into \( p_1 \) (or simply delete \( p_2 \)).

In this example we merge \( B' \) and \( C' \) into \( B \) and \( C \):

\[
\begin{array}{cccccc}
A & \xrightarrow{3} & B & \xrightarrow{4} & C & \xrightarrow{8} & E \\
& & X & & & \\
& & 1 & & 1 & & \\
B' & \xrightarrow{1} & C' & \xrightarrow{2} & & & \\
\end{array}
\]

Then merge \( C''D'' \) into \( C \) and \( D \):

\[
\begin{array}{cccccc}
A & \xrightarrow{3} & B & \xrightarrow{4} & C & \xrightarrow{8} & E \\
& & X & & & \\
& & 1 & & 1 & & \\
B' & \xrightarrow{1} & C' & \xrightarrow{2} & & & \\
\end{array}
\]

And then simplify:

\[
\begin{array}{cccccc}
A & \xrightarrow{3} & B & \xrightarrow{4} & CDE & \xrightarrow{8} & E \\
& & X & & & \\
& & 1 & & 1 & & \\
\end{array}
\]

10.4.4 Removal of erroneous edges

“Chimeric reads” occur when parts of two random DNA fragments fuse together. They give rise to erroneous edges that link nodes that do not belong together.

Such errors cannot be identified based only on graph topology.
However, a simple heuristic is to delete edges of low multiplicity, using a minimum coverage threshold.
Similarly, the topology of a de Bruijn graph can be very tangled due to repeats:
10.4.5 Use of mate pairs

DNA is often sequenced in pairs of reads: 

\[
\begin{array}{cc}
n_{\text{fa}} & n_{\text{fb}} \\
\end{array}
\]

Assume that the insert length has a small variance and most inserts have length \( \leq L \). A node \( v \) in de Bruijn graph is long, if \( \text{length}(v) \geq L \).

**Algorithm:** Mark long nodes that are connected by mate pairs and use to resolve tangles.

**Example:** Assume that \( A \) and \( B \) are long nodes that are joined by mate pairs and that attach to nodes 1, 2, 3, 4, as shown here:

The algorithm modifies the graph so as to obtain the chain \( A, 1, 2, 3, 4, B \):

10.5 Output

The final output of \texttt{Velvet} is generated by identifying all maximal paths in the graph and outputting the corresponding sequences.
10.6 Example

Example of reads and $k$-mers:

\[
\begin{align*}
\text{Reads:} & \quad \text{AATTGCTACGTAGGTATAATATATGACCA} \\
\text{Sequencing:} & \quad \text{Solexa, Illumina, etc..}
\end{align*}
\]

Resulting de Bruijn graph:

\[
\begin{align*}
\text{K-mer graph:} & \quad \text{GATACAATACTAATATCGT}
\end{align*}
\]

Simplified graph:

\[
\begin{align*}
\text{Simplified graph:} & \quad \text{GATAATAATATCGT}
\end{align*}
\]
Detection of tips:

Detection of bubbles:

10.7 Summary

There are two main paradigms for sequence assembly:

- Overlap-Layout-Consensus
- \textit{de Bruijn} graph

Both paradigms are used in current assemblers, for example:

- \textit{Newbler} is an Overlap-Layout-Consensus assembler.
- \textit{Velvet} is a \textit{de Bruijn}-based assembler.