**Department of Computing and Information Systems**

**COMP 90016**

Workshop 9

In this workshop we will see some elementary techniques for *de novo* (i.e. reference-free) transcriptome assembly. The ultimate goal will be to build a toy assembler based upon the notion of Eulerian trails in *de Bruijn graphs.*

De Bruijn Graphs

A *k*-dimensional de Bruijn graph over the alphabet Σ = {A,C,G,T} is a directed graph whose vertex set consists of all possible *k*-mers over Σ. Given an input read of length k + 1, an edge is drawn between the nodes X and Y where X is the suffix of the read and Y is the corresponding prefix.[[1]](#footnote-1)

Example.

*reads* = {AAAT, TACG, GGAA, ATAC}

k=3

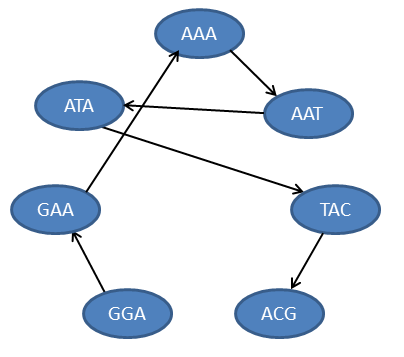


Figure 1 De Bruijn representation set of reads (lone nodes omitted).

Assembly using de Bruijn graphs

Given a de Bruijn representation of a set of *overlapping* reads, genome assembly proceeds as follows:

1. Find a path through through the graph which traverses every edge exactly once, i.e, an *Eulerian trail*.
2. Trace out the sequence defined by the nodes on the path.

In the example above, {GGA -> GAA -> AAA -> AAT ->ATA -> TAC -> ACG} is a possible trail. The assembled genome is GGAATCCG.

**Your tasks in this workshop**.

1.

1. Read in the file TMV.sorted which contains overlapping reads of the tobacco mosaic virus genome sorted alphabetically. Use an appropriate data structure to store the set of reads and confirm that each read is of the same length.
2. Build the de Bruijn representation for the set of reads assuming k = rlen-1.

2. Write a routine to find a path through the graph which traverses each edge exactly once. You may wish to use the included DoubleList which is a python implementation of a doubly-linked list. Output the computed genome.

3 (Discussion).

How realistic is the de Bruijn model for assembly given the type of reads produced by modern sequencing machines? How could the model be modified to accommodate:

1. Errors in reads
2. DNA Repeats
3. Millions of reads? *Billions* of reads?

1. More generally on a read of length *rlen* construct edges between the sequence of rlen + k – 1overlapping *k*-mers. [↑](#footnote-ref-1)