# Genome Assembly – Problems and Solutions

Joseph Christian Somody
BCB410

## Problems

1. To demonstrate understanding of the concepts of genome assembly, write a program in the language of your choice that takes a set of reads as input, utilises a greedy algorithm to assemble the reads, and outputs the assembled sequence. We can assume that the reads are error-free and that there is always full coverage.

2. To demonstrate understanding of how to construct de Bruijn graphs, convert the set of reads below into a de Bruijn graph, where nodes represent strings of length $k - 1$.
   `{GCT/AGC, AGC/GCT, GAG/CTC, CTA/TAG, CGA/TCG}`

3. To demonstrate understanding of the maximum likelihood sequence assembly algorithm proposed by Medvedev and Brudno, discuss two of its shortcomings and how they could be overcome in the future.

# Genome Assembly – Problems and Solutions

Joseph Christian Somody

BCB410

## Solutions

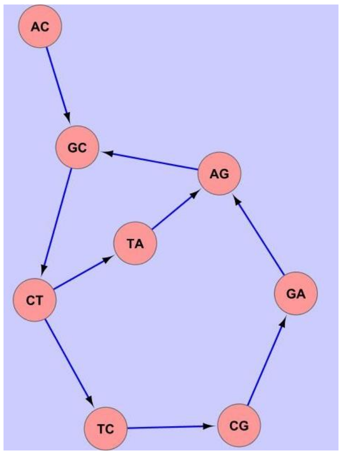
1. Here is an implementation in Python.

```
def score(s1, s2):
    score = 0
    length = min(len(s1), len(s2))
    for i in range(length - 1):
        if (s1[i] != '-') and (s2[i] != '-') and (s1[i] == s2[i]):
            score += 1
    return score

def align(s1, s2):
    max_score = 0
    max_s1 = ''
    max_s2 = ''
    current_score = 0
    current_s1 = ''
    current_s2 = ''
    length = min(len(s1), len(s2))
    for i in range(len(s1) + len(s2) + 1):
        if i < (length):
            current_s1 = ((i + 1) * '-') + s1
            current_s2 = s2 + ((i + len(s1) - len(s2) + 1) * '-')
            current_score = score(current_s1, current_s2)
            if current_score > max_score:
                max_score = current_score
                max_s1 = current_s1
                max_s2 = current_s2
        elif i > (length):
            current_s1 = s1 + ((i - length + len(s2) - len(s1)) * '-')
            current_s2 = ((i - length) * '-') + s2
            current_score = score(current_s1, current_s2)
            if current_score > max_score:
                max_score = current_score
                max_s1 = current_s1
                max_s2 = current_s2
        else:
            current_s1 = s1
            current_s2 = s2
            current_score = score(current_s1, current_s2)
            if current_score > max_score:
                max_score = current_score
                max_s1 = current_s1
                max_s2 = current_s2
    return max_score, max_s1, max_s2

def pairwise(reads):
    max_score = 0
    max_read1 = ''
    max_read2 = ''
    current_score = 0
    current_read1 = ''
    current_read2 = ''
    for read1 in reads:
        for read2 in reads:
            if read1 == read2:
                continue
            current_score, current_read1, current_read2 = align(read1, read2)
            if current_score > max_score:
                max_score = current_score
                max_read1 = current_read1
                max_read2 = current_read2
    return max_read1, max_read2
```

Genome Assembly – Problems and Solutions
Joseph Christian Somody
BCB410

```
def merge(read1, read2):
    merged = ''
    for i in range(len(read1)):
        if (read1[i] != '-') and (read2[i] == '-'):
            merged += read1[i]
        elif (read1[i] == '-') and (read2[i] != '-'):
            merged += read2[i]
        elif read1[i] == read2[i]:
            merged += read1[i]
    return merged

def assemble(reads):
    while len(reads) > 1:
        max_reads = pairwise(reads)
        reads.remove(max_reads[0].strip('-'))
        reads.remove(max_reads[1].strip('-'))
        merged = merge(max_reads[0], max_reads[1])
        reads.append(merged)
    return reads
```

2.



3. Two shortcomings of the algorithm by Medvedev and Brudno are as follows. Firstly, it assumes that reads are error-free. Sequencing is most definitely not error-free, especially with next-next-generation technology. For example, Illumina has specific issues with determining lengths of homopolymer repeats (e.g. CGTTTTTTCA may be reported as CGTTTTTCA or CGTTTTTTTCA). When reads are not error-free, it is not possible to determine overlaps with other reads since there is no insight to that fact in the algorithm. In the future, however, this will become less of an issue. As sequencing becomes faster and less expensive, genomes can be sequenced with more coverage. If an area of a genome is covered 15 times, and there is an error only on one of those 15 reads, it will not matter to the algorithm, since the majority of reads make up for the error (that region can still be joined). Secondly, the algorithm assumes that all regions of the genome are equally sampled. In order for this maximum-likelihood estimate to be accurate, it is assumed that from the superstring most likely to produce the given reads, every area is equally represented (in order to accurately predict copy counts). This, however, is not the case. There will always be regions of a genome that do not amplify well in PCR. Also, single strands may form hairpins. There are many issues that allow for underrepresentation of certain areas of the genome. In the future, sequencing bias scores may be used to correct for this issue, using algorithms that adjust the read frequencies based on the read's sequence.