

Overlap Layout Consensus:

3 stages of OLC are as follows:

1. Overlap:

Based on all pairwise comparisons construct an overlap graph where,

Nodes = Reads (sequences)

Edges = Connections between overlapping reads

2. Layout: look for paths in the overlap graph which are segments of the genome to assemble (contigs)

Goal: find a path visiting all the nodes exactly once

3. Consensus:

following the path, combine the overlapping sequences in the nodes into the sequence of the genome

Source Code / Implementation:

The code was implemented in Python 3 in the Anaconda environment and presented below in a stepwise manner:

Step1: The first step is to delete all occurrences of 'N' in nucleotides

Step2: Return the file size

```
from Bio import SeqIO
filename = "./chr1.fasta"
count = 0
for record in SeqIO.parse(filename, "fasta"):
    print("Record " + record.id + ", length " + str(len(record.seq)))
```

Output:

Record NC_000001.11, length 230481014

...

Step 3: Output first 10K and 100K base pairs for chromosome 1

```
a = None
full_seq = None
for record in SeqIO.parse(filename, "fasta"):
    a = record.seq[:10000]
    full_seq = record.seq[:100000]
print(type(a))
print(len(a))
print(len(full_seq))
```

Output:

<class 'Bio.Seq.Seq'>

10000

100000

Step 4: Generate Random Reads and Count them.

```
import random

def gen_read(seq):
    seq_length = len(seq)
    read_start_pos = random.randint(0, seq_length-501)
    read_len = 400 + random.randint(-100, 100)
    return seq[read_start_pos:read_start_pos+read_len]
b = gen_read(a)
print(b)
print(len(b))
print(type(b))
#count number of reads avg length 400 bp
def count_reads_10X(seq):
    seq_length = len(seq)
    return 10 * seq_length / 400
```

Step 5: Storing the Reads in an object

```
def store_reads(seq, count, archive):
    for i in range(count):
        archive[i] = generate_read(seq)
        if i % 1000 == 0:
            print(str(i) + " / " + str(count))
            archive.dump
```

Step 6: Storing 10K and 100K Reads

```
from klepto.archives import *
archive = file_archive('reads')
store_reads(a, count_reads_10X(a), archive)
len(archive)
archive_full = file_archive('reads_full')
store_reads(full_seq, count_reads_10X(full_seq), archive_full)
len(archive_full)
```

Output:

0 / 2500

1000 / 2500

2000 / 2500

2500

Step 7: Implementing Overlap Algorithm

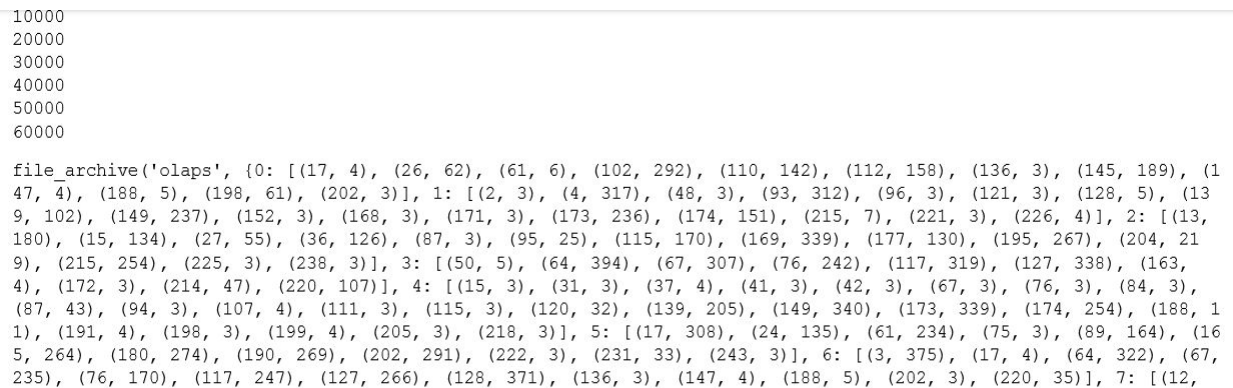
```
def overlap(a, b, min_length=3):
    start = 0
    while True:
        start = a.find(b[:min_length], start) #looks for b's suffix in 'a'
        if start == -1: #no more occurrences to the right
            return 0
        # now checking for full suffix/prefix match
        if b.startswith(a[start:]):
            return len(a)-start
        start += 1
```

Step 8: Generating Overlaps for 10K bp

```
def overlap_map(reads_archive, olap_archive, k):
    i = 0
    for a, b in permutations(range(len(reads_archive)), 2):
        olen = overlap(reads_archive[a], reads_archive[b], min_length=k)
        if olen > 0:
            if not a in olap_archive:
                olap_archive[a] = [(b, olen)]
            else:
                olap_archive[a].append((b, olen))

        i += 1
        if i % 10000 == 0:
            print(str(i))
            olap_archive.dump()
    return olap_archive
#overlaps for 10K bp's
olaps = file_archive('olaps')
overlap_map(archive, olaps, 3)
```

Output: The screenshot is provided here



```
10000
20000
30000
40000
50000
60000

file_archive('olaps', {0: [(17, 4), (26, 62), (61, 6), (102, 292), (110, 142), (112, 158), (136, 3), (145, 189), (1
47, 4), (188, 5), (198, 61), (202, 3)], 1: [(2, 3), (4, 317), (48, 3), (93, 312), (96, 3), (121, 3), (128, 5), (13
9, 102), (149, 237), (152, 3), (168, 3), (171, 3), (173, 236), (174, 151), (215, 7), (221, 3), (226, 4)], 2: [(13,
180), (15, 134), (27, 55), (36, 126), (87, 3), (95, 25), (115, 170), (169, 339), (177, 130), (195, 267), (204, 21
9), (215, 254), (225, 3), (238, 3)], 3: [(50, 5), (64, 394), (67, 307), (76, 242), (117, 319), (127, 338), (163,
4), (172, 3), (214, 47), (220, 107)], 4: [(15, 3), (31, 3), (37, 4), (41, 3), (42, 3), (67, 3), (76, 3), (84, 3),
(87, 43), (94, 3), (107, 4), (111, 3), (115, 3), (120, 32), (139, 205), (149, 340), (173, 339), (174, 254), (188, 1
1), (191, 4), (198, 3), (199, 4), (205, 3), (218, 3)], 5: [(17, 308), (24, 135), (61, 234), (75, 3), (89, 164), (16
5, 264), (180, 274), (190, 269), (202, 291), (222, 3), (231, 33), (243, 3)], 6: [(3, 375), (17, 4), (64, 322), (67,
235), (76, 170), (117, 247), (127, 266), (128, 371), (136, 3), (147, 4), (188, 5), (202, 3), (220, 35)], 7: [(12,
```

Step 9: Generating Overlaps for 100K bp's

```
olaps_full = file_archive('olaps_full')
temp = overlap_map(archive_full, olaps_full, 3)
```

Output:

```
200000
300000
310000
320000
330000
340000
350000
360000
370000
380000
390000
400000
410000
420000
430000
440000
450000
```

Step 10: Finding the Longest Path in the Graph

```
from collections import defaultdict

def exhaustive(graph, startnode, endnode):
    maxdist = -1
    stack = [[startnode], 0]
    while stack:
        cpath, cdist = stack.pop()
        cnode = cpath[-1]
        if cnode == endnode:
            if cdist > maxdist:
                maxdist = cdist
                maxpath = cpath
            continue
        for nbr, nbrdist in graph[cnode]:
            stack.append( (cpath+[nbr],nbrdist+cdist) )

    return maxdist,maxpath

def max_node(node):
    max_node_dist = -1
    max_node = 0
    try:
        for x, y in node:
            if max_node_dist < y:
                max_node_dist = y
                max_node = x
    except:
        print "exception"
    return max_node, max_node_dist

def greedy(graph, startnode):
    maxdist = 0
```

```

adjacent = [(startnode, 0)]
adj_old = adjacent[:]
maxpath = []
while adjacent:
    max_node_, max_node_dist = max_node(adjacent)
    if max_node_dist == -1:
        break
    if max_node_ not in maxpath:
        maxdist = maxdist + max_node_dist
        maxpath.append(max_node_)
        adjacent = graph[max_node_]
    else:
        break

    if adj_old == adjacent:
        break

    return maxdist, maxpath

if __name__ == "__main__":

    graph = {0:[(1, 935.5), (2, 147297.5)], 1:[(3, 1e-10), (4, 945.8)],
             2:[(3, 1e-10), (4, 945.8)], 3:[(5, 3656)], 4:[(6, 7669.5), (7, 18500.5)],
             5:[(6, 7669.5), (7, 18500.5)], 6:[(8, 100)], 7:[(8, 100)], 8:[]}
    startnode, endnode = 0, 8
    maxdist, maxpath = greedy(graph, startnode)
    print("Maxpath is %s, Maxdist is %d" % (maxpath, maxdist))

    graph = {0:[(1, 5), (2, 3)], 1:[(3, 6), (2, 2)],
             2:[(4, 4), (5, 2), (3, 7)], 3:[(5, 1), (4, -1)],
             4:[(5, -2)], 5:[]}
    startnode, endnode = 0, 5
    maxdist, maxpath = greedy(graph, startnode)
    print("Maxpath is %s, maxdist is %d" % (maxpath, maxdist))

```

Output:

Maxpath is [0, 2, 4, 7, 8], Maxdist is 166843

Maxpath is [0, 1, 3, 5], maxdist is 12

Step 11: Longest path for 10K Base Pairs

```

global_max = ([], 0)
for startnode in olaps.keys():
    maxdist, maxpath = greedy(olaps, startnode)
    #print("Maxdist is %d, maxpath is %s" % (maxdist, maxpath))
    if global_max[1] < maxdist:
        global_max = (maxpath, maxdist)
print("Global maxdist is %d, maxpath is %s" % (global_max[1], global_max[0]))

```

Output:

```
Global maxdist is 46973, maxpath is [35, 83, 7, 138, 119, 97, 170, 141, 144, 206, 146, 5, 17, 180, 89, 24, 231, 101, 224, 11, 84, 111, 32, 45, 203, 211, 155, 217, 8, 179, 113, 240, 96, 232, 21, 131, 80, 196, 14, 185, 237, 243, 46, 20, 47, 247, 71, 29, 106, 49, 109, 169, 195, 13, 15, 36, 27, 95, 39, 209, 63, 53, 105, 6, 3, 64, 117, 67, 220, 214, 0, 10, 2, 145, 112, 26, 31, 248, 171, 38, 125, 225, 172, 107, 140, 142, 41, 161, 134, 239, 65, 200, 158, 197, 241, 234, 104, 176, 218, 193, 68, 186, 9, 205, 194, 199, 207, 153, 74, 150, 77, 88, 151, 168, 116, 73, 132, 163, 192, 4, 149, 139, 8, 7, 159, 143, 184, 154, 219, 238, 108, 230, 23]
```

References:

1. Wikipedia- Overlap Layout Consensus
2. Miller JR, Koren S, Sutton G. Assembly Algorithms for Next-Generation Sequencing Data Genomics. 2010;95(6):315-327. doi:10.1016/j.ygeno.2010.03.001.
- 3.<http://www.cs.yale.edu/publications/techreports/tr1470.pdf>
- 4.<https://pypi.python.org/pypi/klepto>